Modeling and analyzing a generic nine node traffic network using a microscopic simulator

Vivek Barthwal

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.07.2020

Supervisor and advisor

Prof. Themistoklis Charalambous

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Vivek Barthwal

Title Modeling and analyzing a generic nine node traffic network using a microscopic simulator

Degree programme Erasmus Mundus Spacemaster

Major Space robotics and automation **Code of major** ELEC3047

Supervisor and advisor Prof. Themistoklis Charalambous

Date 31.07.2020 **Number of pages** 34+46 **Language** English

Abstract

Traditionally traffic signals are controlled by a fixed cycle algorithm. However, in the urban transportation network which is well defined this can be bettered using adaptive signalling.

The thesis examines a generic nine node network to verify the efficiency of Max-weight algorithm under different scenarios. Traffic simulations are achieved using PTV Vissim 11, a microscopic traffic simulator, whereas MATLAB R2019b was used to execute the algorithm and plot the results. The communication between the two software requires parallel computation of multiple lanes at any particular moment. The interface was studied to optimise transfer of vehicular information.

The analysis confirmed our belief as Max-Weight algorithm was more efficient than the fixed cyclic control. Even the addition of higher levels of noise outperformed the network running on a fixed cycle algorithm. However, a filtered max-weight algorithm is theoretically the most pragmatic solution for traffic control. This is discussed but not implemented in the report and is definitely a topic to investigate in the future.

Keywords Traffic networks, Traffic control, Max-weight algorithm, Back-pressure algorithm, Parallel computation, Particle filter

Preface

I want to thank Professor Themistoklis Charalambous and for his good guidance.

Otaniemi, 31.07.2020

Vivek Barthwal

Contents

Symbols and abbreviations

Abbreviations

MW	Max weight
F	Phase F: free-flow phase
C	Phase C: congested traffic flow phase
S	Phase S: synchronized phase
J	Phase J: wide moving jam phase
OD	Origin-Destination

1 Introduction

Traffic congestion in urban settlements leads to a monetary loss of human potential, health deterioration and environmental damage. The segregation of residential and work areas results to an uneven distribution of vehicle flow. There are obvious differences between urban centres across the world. In cities that had a sudden economic boom or influx of people, the urban infrastructure often isn't equipped with the right resources. However the cities that have seen gradual economic growth alongside regulated population increase have a well defined structure that can be mapped and in turn can lead to a controlled traffic flow.

The usual way of regulating traffic is by using fixed cycle traffic signal controllers. This implies that at road intersections, there exists a predefined order to switch between the lanes. But in cities like 'Helsinki' where we have the vehicle data available for every lane, there is a possibility to model signal controllers in an adaptive manner. An optimal approach that has been used in previous research was of max-weight algorithm which provides a method to dynamically route traffic that leads to a maximum throughput network. The algorithm is affected by noisy measurements, hence the particle filter is used to test the network.

In order to apply the algorithm there exists a need of parallel computation between multiple lanes within an intersection. The data is gathered via PTV Vissim, a microscopic traffic simulator, along with MATLAB 2019b to execute the algorithm and to apply the filter.

The thesis is organized as follows. In section 2, the theoretical part of the project as well as the methods and the algorithms used, are explained. In section 3, we discuss the communication between the two software to achieve parallel computation of multiple lanes simultaneously. Further, we applied the Max-Weight algorithm for our nine node network and later added noise to replicate real life environment. We came to the conclusion that addition of higher levels of noise can bring down the algorithm's efficiency. This reinstated the need of particle filtering, which is discussed theoretically in section 2. The results are presented and analyzed in 4. Finally, in section 5, conclusions are drawn to pave the path for future developments.

2 Background

2.1 Microscopic and macroscopic traffic flow modelling

Traffic flow in an organised urban settlement can be analyzed via various models. In this thesis, we focus on microscopic to simulate the actual network while the macroscopic is used to develop the model.

2.1.1 Microscopic traffic simulation

Microscopic traffic models simulate single vehicular units. In other words, it shows minute details of traffic flow and the interaction between these units. This can be further divided into discrete and continuous. The discrete model is called cell automata, a network formed by vehicles that move between cells in discrete steps. This method is often unrealistic as the division of cells and vehicle can only be an approximated one. However, the continuous model mimics reality and is more complex. The software used, PTV Vissim comes under this category.

The continuity of the model can again be governed by two approaches. One where the vehicular data is governed by the safe distance between other vehicles and the other where the model is based on the desired speed of a vehicle.

data.png

Data Collection Results

Select layout...

Count: 3172	SimRun	TimeInt	DataCollectionMeasurement	Acceleration(All)	Dist(All)	Length(All)	Vehs(All)	Pers(All)	QueueDelay(All)	SpeedAvgArith(All)	SpeedAvgHarm(All)	OccupRate(All)
1	1	0-60	1: Porkkalankatu	-0.51	43.05	4.20	3	3	0.00	47.80	47.74	18.59 %
2	1	0-60	2: Ruoholahdenkatu 16				0	0				0.00 %
3	1	0-60	3: Ruoholahdenkatu 6				0	0				0.00 %
4	1	0-60	4: Lapinlahdenkatu 15				0	0				0.00 %
5	1	0-60	5: Malminrinne 3	-1.95	35.90	4.20	2	2	0.00	47.92	47.91	12.94 %
6	1	0-60	6: Albertinkatu 44 N				0	0				0.00 %
7	1	0-60	7: Albertinkatu 44 S				0	0				0.00 %
8	1	0-60	8: Mechelininkatu S	-2.11	44.76	4.01	1	1	0.00	47.21	47.21	6.28 %

Figure 1: Vehicle data transmitted via VISSIM through interface

2.1.2 Macroscopic traffic simulation

Macroscopic traffic model considers vehicles as particles flowing in the road, similar to liquid and gas.

The evaluations were done by monitoring every single vehicular data such as speed and travel time using our microscopic model. Even the queue calculation was done by monitoring the speed of every single vehicle at every possible time step (here, 1s). All these parameters are then integrated using a macroscopic model which are related by the equation:

$$q(x,t) = \rho(x,t)v(x,t)$$

The three variables flow rate(q), flow density(ρ) and vehicle velocity(v) are all defined for a fixed point 'x' and time 't'.

2.2 Three-phase model

2.2.1 Fundamental diagram of traffic flow

The classical theories of traffic flow have two distinct phases: free flow (F) and congested traffic (C). They are represented via a fundamental diagram of traffic flow.

The diagram relates the traffic flow (vehicles/time unit) with the traffic density (vehicles/distance unit) as dictated by the macroscopic traffic model equation. Since our network is generic and has defined set of link lanes combination, the distance unit can be written off as constant. Thus, the fundamental diagram is drawn with the parameters: vehicular flow (vehicles/min) and vehicular density.

The free flowing branch of traffic flow is shown with the positive line slope, and the congested branch is shown with a negative slope. The traffic flow increases with the increase of traffic density until it reaches the maximum point of traffic flow ρ_c, which is called the critical density. From this point on, the higher the density on the congested branch, the lower the traffic flow.

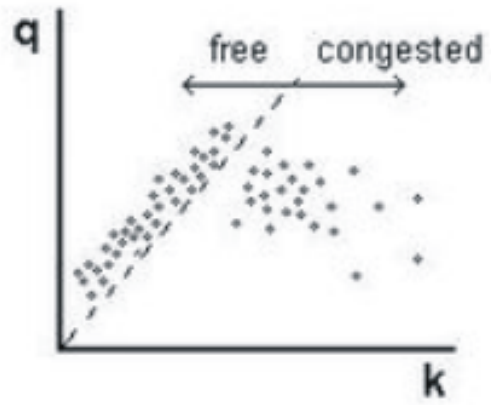


Figure 2: Classical theory: fundamental diagram

F phase: $q \propto \rho$; minimum velocity is denoted with v_{free}^{min}.
C phase: $q \propto \frac{1}{\rho}$; velocity at all points, $v_{congested} < v_{free}^{min}$.

2.2.2 Three-phase traffic theory

Three phase theory is an upgraded explanation of the physics of traffic breakdown and resulting congested traffic on highways. It divides the congested branch into two other parts, synchronised flow S and the wide moving jam J.

As described in the previous section, C phase has at all times lower velocity than the F phase. The subdivision hence occurs on the flow rate, if the flow is maintained it is classified under phase S otherwise phase J. The decrease in velocity is compensated with the increase in traffic density under phase S while in phase J, the speed decreases drastically and the traffic density reaches its maximum (a constant value).

There are two transitions that occur in this model, F to S and S to J. F to S happens at the critical density point. It usually occurs with random fluctuations of

the traffic due to vehicle deceleration, lane changing or erratic driving behaviour. A wide jam gradually happens from the synchronised flow with which there is a drop in speed of vehicles. In this thesis, the three phase model will be used for traffic analysis.

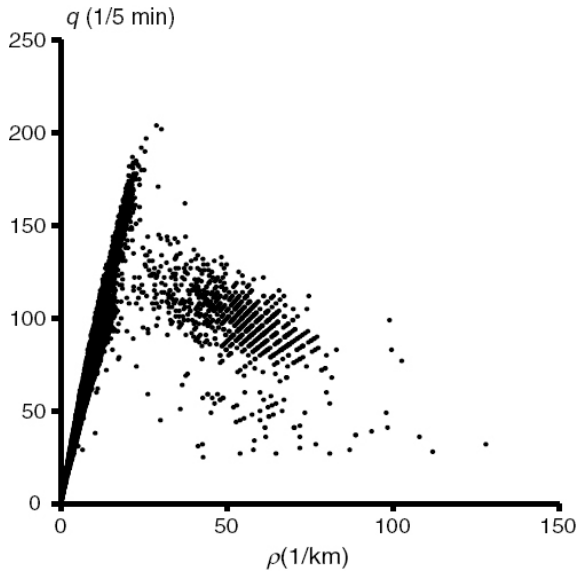


Figure 3: The typical fundamental diagram (the relation between vehicle density and flow rate) from 1 month of data measured at a point on a freeway. The critical density is nearly 25 (vehicles/km). The data were measured by the Japan Highway Public Cooperation. [6]

2.3 Max-weight algorithm

The max-weight algorithm, also known as the back-pressure algorithm, is a method for dynamically routing traffic over a queuing network that leads to maximum network throughput. It was first proposed in the seminal paper [7] by Tassiulas and Ephremides, and since then, it has been used and enhanced, mostly in communication network applications. However, in this thesis, the algorithm is applied in transportation systems. This work implements the solution proposed in [8], and it considers decentralized traffic signal control policies. The paper demonstrates that the standard Max-Weight algorithm is optimal even under noisy queue measurements, and the use of filtering improves the performance of the algorithm under the same conditions. In this project, we will test and analyze the algorithm under different circumstances.

2.3.1 System model

The model that we are using is a generic model with 9 junctions. Each one is a 4 way junction which constitutes of 8 roads and 12 possible traffic movements. It is mathematically represented via a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, also known as standard road network model. The term i, represents the lanes of the roads, which are capable

of queuing vehicles, and the term $\varepsilon_{ij} \in \mathcal{E}$ represents the possible flows or paths from the lane i to lane j. Lane i is known as the outgoing lane, and lane j is also called an incoming lane. Roads (or links as per VISSIM) can have multiple lanes and each one of them can be individually accessed for information retrieval.

In the set of diagrams below we see one junction in isolation along with the possible pathways that can originate from it.

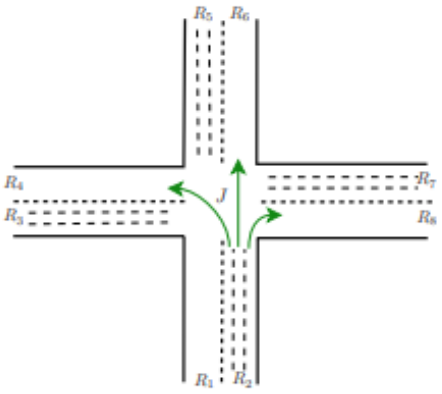


Figure 4: A four-way junction [11].

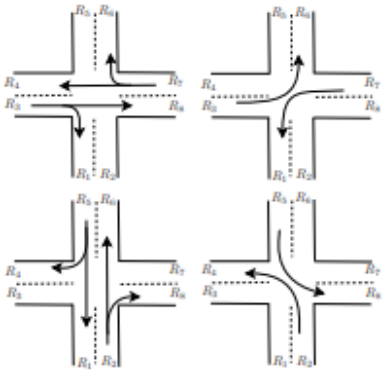


Figure 5: Typical phases [11].

It is assumed that decisions in a junction are taken synchronously and that all the controllers at the intersections have the same cycle of length T. The time (T) is considered as discrete and idle times during switches are not taken into account. In entirety, all the junctions should ideally have no delay too but the lack of parallel decision making capability makes the model erroneous. The error however is infinitesimal in a set of just 9 junctions.

2.3.2 Services rates and traffic phases

During the simulation, the number of vehicles served at time t, from the selected node i to its out-neighbors j is denoted as $S_{ij}(t)$. Each intersection has a signal controller, which in turn has signal groups allocated to it. Each signal group has a finite set of signal heads attached to it that changes their sequences simultaneously. This sequence of signal group activation is hence a collection of phases P_l. For each phase $\sigma \in P_l$, a set of non-overlapping links are activated at every slot $t \in N$, and a number of vehicles $S_{ij}(t)$ are transferred by lane i to lanes j. The activation of a concrete phase is decided by the Max-Weight algorithm.

2.3.3 Pressure release and phase choice

The algorithm is designed to make decisions in order to minimize the sum of squares of queue backlogs in the network from one time-slot to the next. It chooses transmission and routing variables to greedily minimize a bound on the drift of a Lyapunov function [9][10]. Specifically, the current queue of a node i at the beginning time slot is denoted as $Q_i(t)$. For each lane i going to lane j, the pressure weight W_{ij} is

computed as follows:

$$W_{ij}\Big(Q(t)\Big) = Q_i(t) - Q_j(t)$$

For each phase $\sigma \in P_l$, the junction computes the pressure release

$$w_\sigma\Big(Q(t)\Big) = \sum_{(i,j)\in\sigma} W_{ij}\Big(Q(t)\Big) S_{ij}(t), \forall\sigma \in P_l,$$

and chooses the one with the highest pressure release, i.e.,

$$\sigma^*(t) = \arg\max_{\sigma\in P_l} w_\sigma\Big(Q(t)\Big)$$

In this thesis it is considered that the service rate is the same for all the lanes. In that case, the pressure release can be computed as follows:

$$w_\sigma\Big(Q(t)\Big) = \sum_{(i,j)\in\sigma} W_{ij}\Big(Q(t)\Big),\ forall\sigma \in P_l,$$

2.3.4 The Noisy-Max-Weight (or Noisy-Back-Pressure) algorithm

The noisy measurement of queue backlog (i.e., the number of cars) of a node i is denoted as $\widetilde{Q}_i(t)$, and $\nu_i(t)$ is the aforementioned noise at lane i at a time t.

$$\widetilde{Q}_i(t) = Q_i(t) + \nu_i(t)$$

Since it is considered that queues are finite and positive, $\nu_i(t)$ is finite for all i. For each lane i going to lane j and for the case where the queue length measurements are noisy, the pressure weight W_{ij} computation is done as follows:

$$W_{ij}\Big(\widetilde{Q}(t)\Big) = \widetilde{Q}_i(t) - \widetilde{Q}_j(t)$$

After this point, the pressure release and phase choice is made as in the case mentioned above, using the noisy pressure weight $W_{ij}\Big(\widetilde{Q}(t)\Big)$.

2.4 The filtered max-weight algorithm

In conjunction with the results provided by the queue state estimation along with the noise, we can add a filter that helps determine the exact value of queue which can then be passed on to the maximum weight algorithm.

For this, the suggested method is of using a Bayesian sequential estimator. In simple words, we would try to predict the queue state of the link ($Q_i(t+1)$) based on the queue state at a prior time interval($Q_i(t)$), queue state of other incoming lanes($Q_{j\in\mathcal{N}_i^-}(t)$) and a random variable called arrival rate($A_i(t)$). In our program, we have used inflow rate and have simulated the number of vehicles in a queue instead. The process and measurement noise are independent, identically distributed stochastic processes, but not necessarily additive or Gaussian. The initial state of the queue is independent of the noise processes and its distribution is given through a probability density function, p(Q(0)).

$$Q_i(t+1) = f_{a_i(t),a_{jcN_i}(t)}\left(Q_i(t), Q_{j\in\mathcal{N}_i^-}(t), A_i(t)\right)$$
$$\widetilde{Q}_i(t) = Q_i(t) + \nu_i(t)$$

Our system is non-linear with discrete queue states and the noise processes are non-Gaussian and bounded. Based on all these parameters, the preferred models are the ones implementing hidden Markov or the implementation of particle filter. Paper [8] advocates the use of distributed particle filter with belief propagation network.

Belief propagation involves two steps: prediction and correction. For traffic flow network, paper [3] suggests a rather straightforward approach which can be briefly explained as follows:

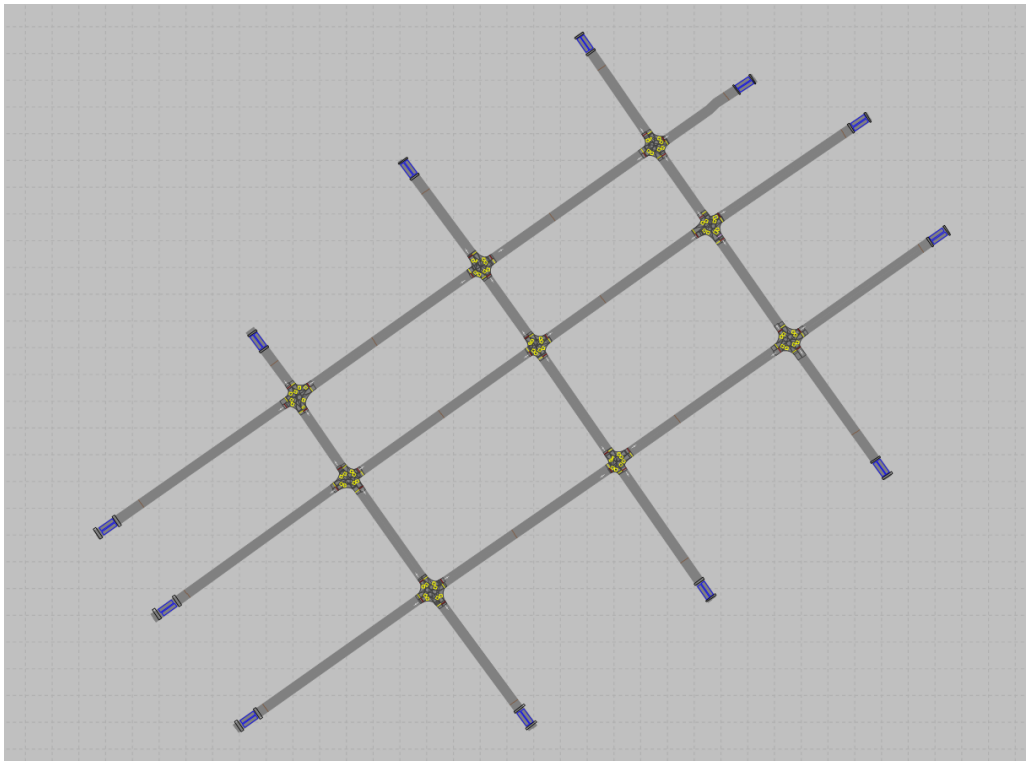
- The posterior probability of finding an estimated value of queue at any lane in a given moment is dependent on the noisy queue measurements and the noise levels.
- The latter two helps us locate the likelihood of the queue length (or, number of vehicles).
- Based on this, particles can be distributed. Using importance sampling, a manner by which we can generate stationary probability distribution, we can reduce the domain to its subset.
- Further re-sampling can be done by using multinomial distribution and randomly selecting a weighted particle. This is where the randomness comes and has to be solved.

The succinct explanation is important to steer our focus towards re-sampling. Due to random picking of weighted particles, there is a chance that the particle selected might not be the best choice. So, going further its implementation requires additional environmental data to help us prioritise the weight distribution better. The thesis doesn't use particle filter algorithm further and is just a comparison of the cyclic control with max-weight algorithm (with and without noise in the environment).

3 Research material and methods

3.1 Signal controller for a generic network

We have computed all our parameters over a network of nine intersections. The network is modelled on real section of network in Kamppi area but has a distinction, it has a signal controller at every intersection. The word signal controller tells the scheme of operation of a single junction comprising of 8 link lane combination. Since the focus was simply on a generic network of just nine lanes, there's no parallel transmission of data between controllers when computing the length of queues. The network and an individual intersection point are shown below.



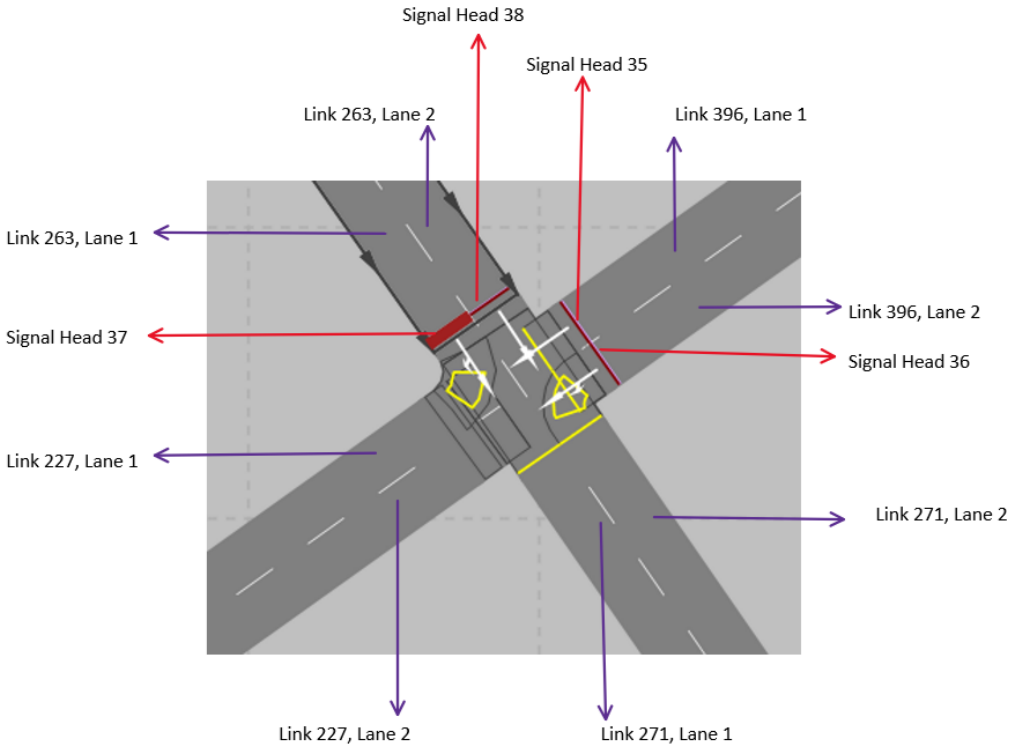


Figure 6: Link, lane and signal heads under a signal controller

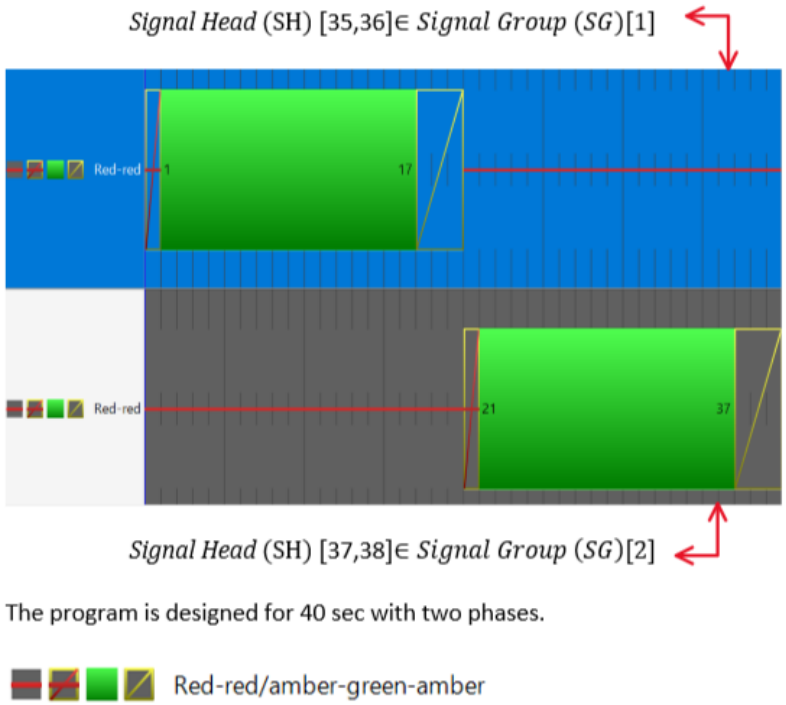


Figure 7: Signal head states

The traffic demand was estimated based on the traffic data published by Helsinki

Urban Environment Division[13] and further adjusted with real-time traffic from Google Maps[14] over a selected area. The dynamic assignment for evaluating vehicle inputs on a certain link was achieved by employing origin-destination matrices[12] using PTV Vissim. The number of zones are the starting and end points for every individual vehicle.

3.2 Simulation

3.2.1 Approach towards case implementation

To run the simulation, the lanes were predetermined out of all the possible ones being computed at a specific instance of time. As multiple lanes' data was computed simultaneously, the approach taken was of parallel computation. The interface containing the vehicular data of a specific link was a dynamic value serially transferred.

The deserialization wasn't achieved because the object was to be computed every second. The error caused was due to dot indexing in the communication interface. Dot indexing for broadcast variables, the variables that are being distributed for every iteration in a parallel for loop, gives an error as the process isn't allowed in MATLAB. The links and lanes are hence replaced by storing the values in a cell or a matrix and only the real dynamic vehicular data was called upon during parallel computation. This lead to the link's lane interface successful distribution. The broadcasting was hence achieved, but further interruption was caused when the vehicles were called out for every lane. This was due to the class change for the object in every iteration from interface to double that was just displayed as an empty matrix or cell.

Further analysis for parallel computation led to the process of single program, multiple data (SPMD) technique. It runs on multiple processors and can be divided between the employed workers equally. Since we need an every instance computation, the sample space for our lanes being computed reduced the sample space to the number of workers. Thus it wasn't as effective as parallel for loops in comparison. These parallel divisions when called by their indexes again lead to class change from interface to double displayed as empty matrix or cell.

Because of the above mentioned issues, we shortened the network to a set of nine controllers. The controllers can be controlled with minute errors as the computation isn't as extensive as doing it over a whole city network.

So for our nine signal controller interaction network, we structure our simulations under four cases. The first case(1) corresponds to fixed and cyclic signal controllers, where they switch phases without decisions. Other three cases(2,3,4) employ the Max-weight algorithm. For case 3 and 4, noise is added to the queue (10% and 25%). Thus, we further divide our cases 3 and 4 as 3(10),3(25),4(10) and 4(25). Percentages here is the amount of noise added in the queue measurement to replicate

real case scenarios. This is done by measuring the length of the lane of a particular link and then dividing it by the length of an individual vehicle (5 metres). This tells us approximately the number of vehicles that can fit in the particular link. Now 10% and 25% is just the number of vehicles that are added to the queue formed at a particular lane during a particular instant. And finally, we try to apply a particle filter over our noisy queue measurements in cases 4(10) and 4(25). They are all listed in the tabulated form below.

Case 1	Fixed & cyclic signal head
Case 2	Maximum weight algorithm
Case 3(10), 3(25)	Maximum weight algorithm, 10% or 25% noise level
Case 4(10), 4(25)	Maximum weight algorithm, 10% or 25% noise level, particle filter

Table 1: Case explanation

Our expected efficiency of the cases can be expressed logically as :

$$Case4(10) > Case3(10) > Case2 > Case1$$

$$Case4(25) > Case3(25) > Case2 > Case1$$

Only the case of 4(10) and 4(25) are the ones that have shown inconsistent results in prior research. In continuation with the same model, we implement all four cases for our road network of two signals. Each case was simulated with seven inflow values, in order to gather an adequate amount of data. Inflow values are just different inflow multipliers applied over an OD matrix. OD matrix is a list of vehicles origin and destination based on the zones assigned by us, this can be accessed directly using PTV Vissim. Thus, 42 simulations of one hour each are carried out (7 inflow values, 6 cases) to reach steady state observations.

The measurements are after every 5 seconds and signal heads phases were switched every 20 seconds. This gives a total of 720 measurements every hour and the impact of switching is visible with this time interval. The simulation resolution was set at 1 second just to mimic the previous research three cases. The resolution in this case is a value that tells us how various entities, such as pedestrians and vehicles, would interact with each other within the network. But since we are just communicating between a set of nine signal controllers we have a limited combination of link and lanes in our network. Thus for such selection of link-lanes we have to compute the sum of departed vehicles from their origin. This can be computed in total but not in isolation. So, we track the vehicles by their unique identification number and evaluate the total number of vehicles originating from a specific zone. Apart from this, the zones of their origin and destination can be mapped with the number of vehicles in that instant. This is helpful especially when we measure other pairs of intersections. This can help us sift through unnecessary zone details as they wouldn't be a part of our simulations. Two mat files were used to import the data to reduce an already excessive computation.

3.2.2 Methodology

The communication between PTV Vissim and MATLAB is achieved via Vissim's COM interface. The interface enables inter-process communication object creation that was used to launch the simulation, modify parameters and access data.[15][16]

There are four MATLAB scripts for each of the four primary cases. For the subdivisions in cases 3 and 4, the noise level is the only varying factor which was integrated in the same script. A set of variables are checked per lane in every simulation: queue, speed, unique ID, time in network, etc. Once all of this data is collected, the maximum weight algorithm is run to find the best suited signal head in the two signal controllers at the two intersections.

There are four entities measured over the unique ID vehicles present in the road intersection: average queue length, average speed, average travel time and the sum of departed vehicles. Apart from these, queue length for the last computed simulation of each case (inflow value: 2.5) and fundamental network diagram for all the cases are plotted too. The values are stored and superimposed to show a fair comparison in all these cases.

4 Results

4.1 Network fundamental diagram and queue length evaluation

4.1.1 Saturated network (inflow rate=3)

In order to understand these results we have to familiarise ourselves with network configuration and network specific terms. As the network is really small, we reach a point of saturation in vehicular flow. To not reach this point at any instance would be our ideal case. In order to evaluate how these cases perform at the peak point, we will evaluate it for an inflow rate of 3.

The terms used for plotting are 'queue', 'queue sum' and 'queue mean'. Let's go through them for our network one by one.

Term used	Explanation	Frequency	Size of matrix
Queue	Number of vehicles in a lane	Every 5s	[48*720]
Queue sum	Number of vehicles at an instant (all lanes)	Every 5s	[1*720]
Queue mean	Mean of vehicles present in all the lanes during a particular inflow	Once hourly	[1]

Table 2: Queue terms and their usage

We use these terms in conjunction with other terms to better understand and compare each model. The network flow diagram , as explained in three phase model structure above, has three stages: free flow, saturated wide moving jam and congested branch. It is evaluated over the number of vehicles that have reached their destination (in our terms, destination zone) and the number of vehicles that are present in the network at that instance (flow rate of vehicles). We will see that this curve is diligently followed in almost all models. Though, with a smaller network all the models saturate. So, the number of vehicles in all these models are not that distinguishable aka total number of vehicles reaching their destination is 400-450 for all. Hence, the prominent features to be noted in our case is:

- Time taken for saturation: As expected, the fixed signal system saturates fastest, thus the flow rate has a sudden and abrupt stop as early as 2745 seconds. In all other cases, saturation is reached after 3000 seconds.
- Critical density: The point where maximum flow rate is achieved in a system. In our case, it happens in case 2, meanwhile in case 3(25) the graph becomes extremely erratic. The flow rate doesn't decrease which shows that the vehicles haven't saturated even when other models have. This can further be confirmed by the adjoining queue sum graph. As the noise addition can cause vehicular flow to increase or decrease at any point, this is a favourable outcome by chance.

- Smoothness of the curve: We need an even distribution for the network where increase or decrease are gradual as compared to sudden. If the curve is near it's critical density it implies that the flow rate is around it's maximum capacity for a fair amount of time. The mid points of queue sum over time is measured across all three cases and to further demarcate the even distribution of vehicles we have drawn their normal probability density functions. In this aspect, case 2 stands out as it has an almost ideal curve. Case 3(25) is erratic but 3(10) fares well in this aspect.

Evaluation of case 1

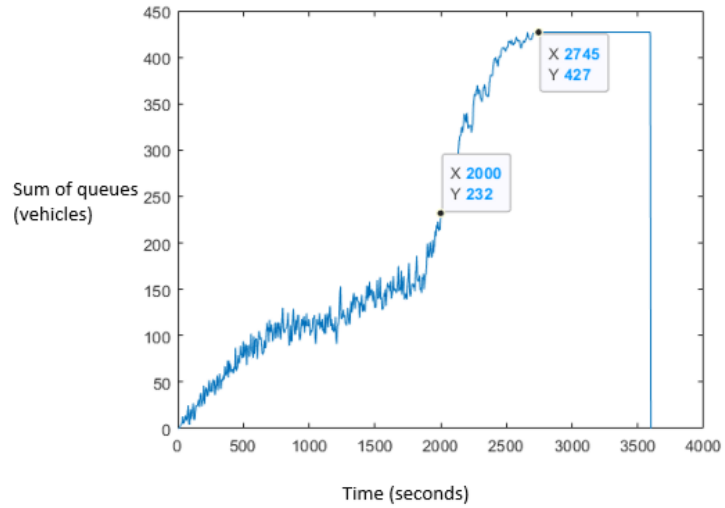


Figure 8: Queue length

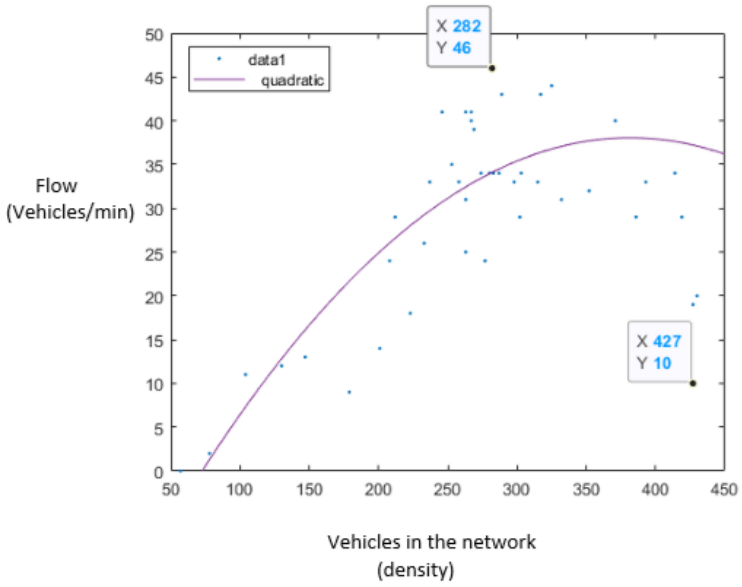


Figure 9: NFD with curve fitting

Evaluation of case 2

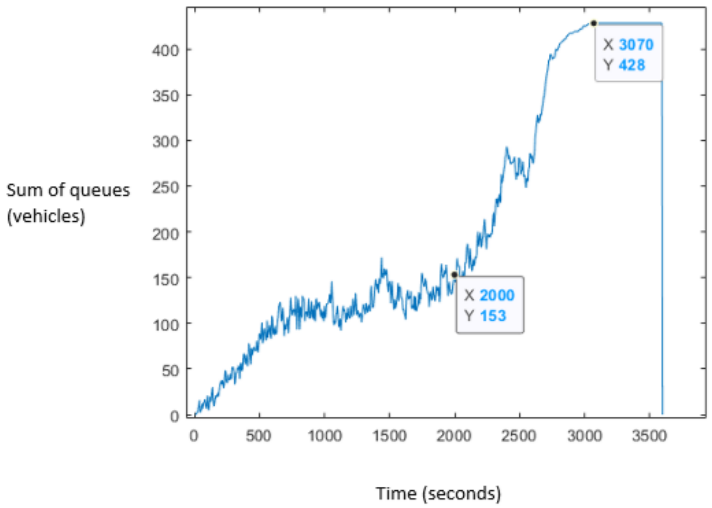


Figure 10: Queue length

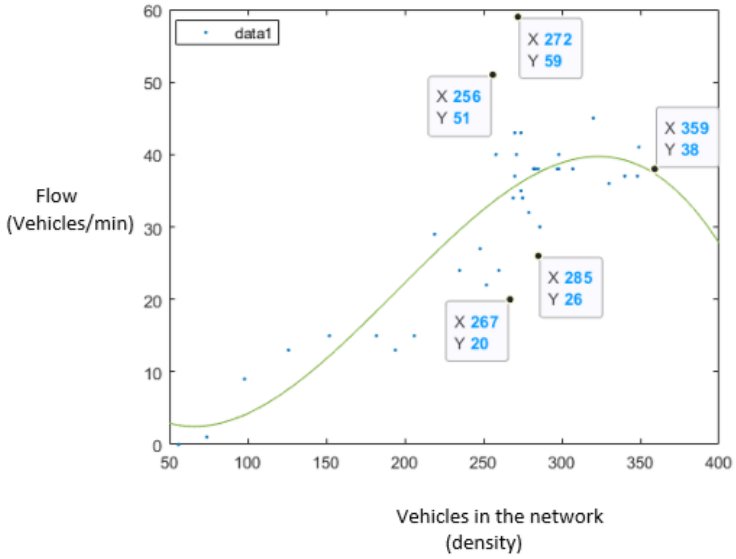


Figure 11: NFD with curve fitting

Evaluation of case 3(10)

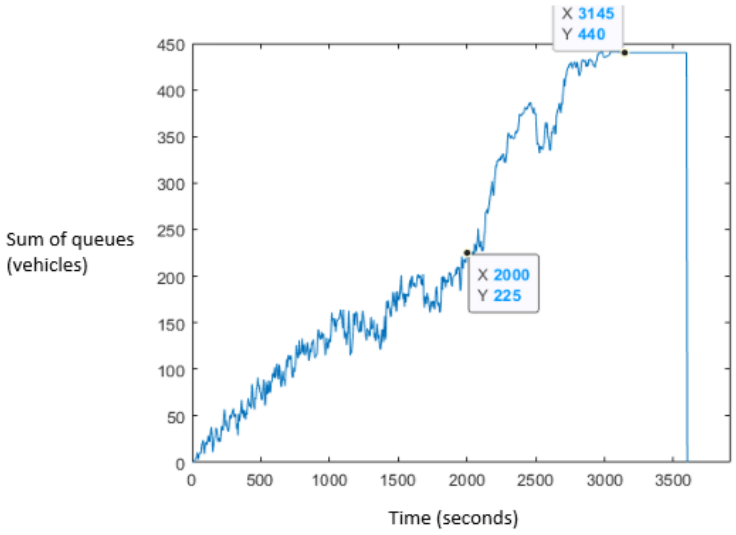


Figure 12: Queue length

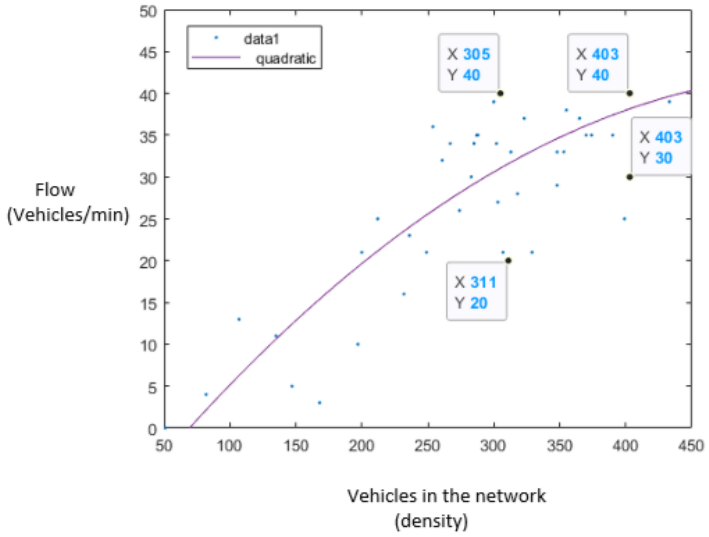


Figure 13: NFD with curve fitting

Evaluation of case 3(25)

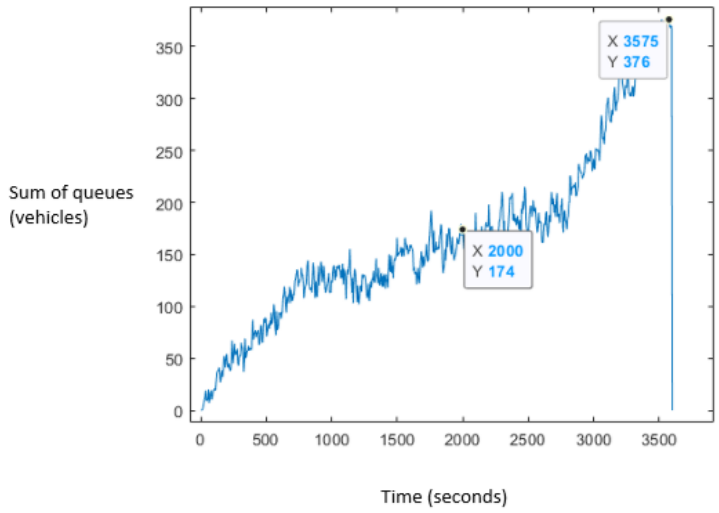


Figure 14: Queue length

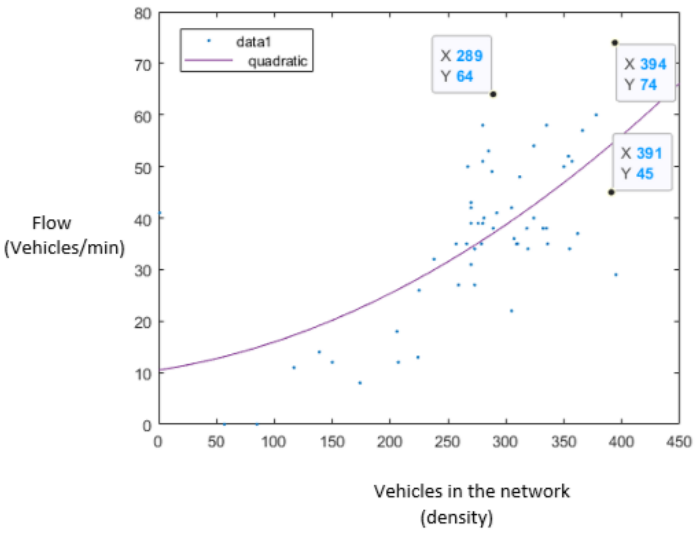


Figure 15: NFD with curve fitting

We have drawn the normal distribution of the mean of queues to see how even the distribution is. The comparison excludes the case of 25% noise as it's visibly uneven from the NFD itself. Here, we see that the peak is reached in the case 2 as expected. This further confirms our belief that the vehicles would operate at or around the critical density point for the maximum amount of time in case 2.

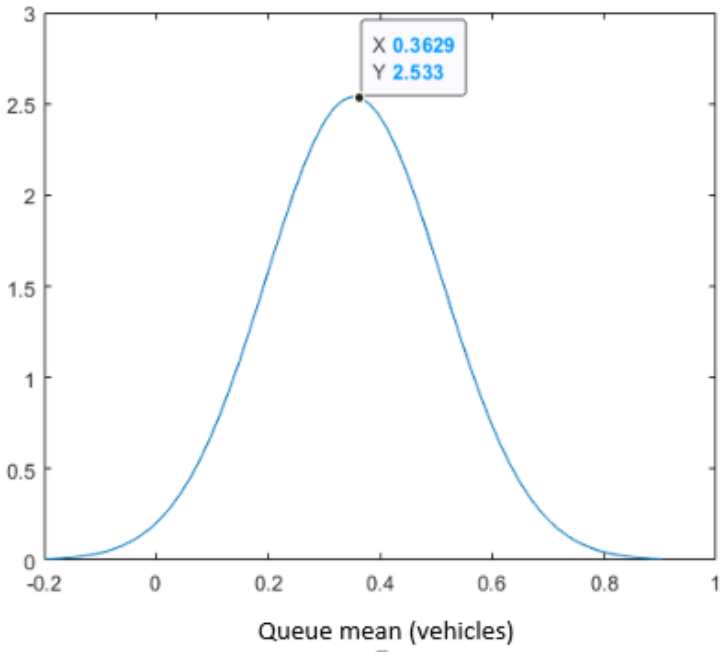


Figure 16: Normal distribution of the mean of queue length (case 1)

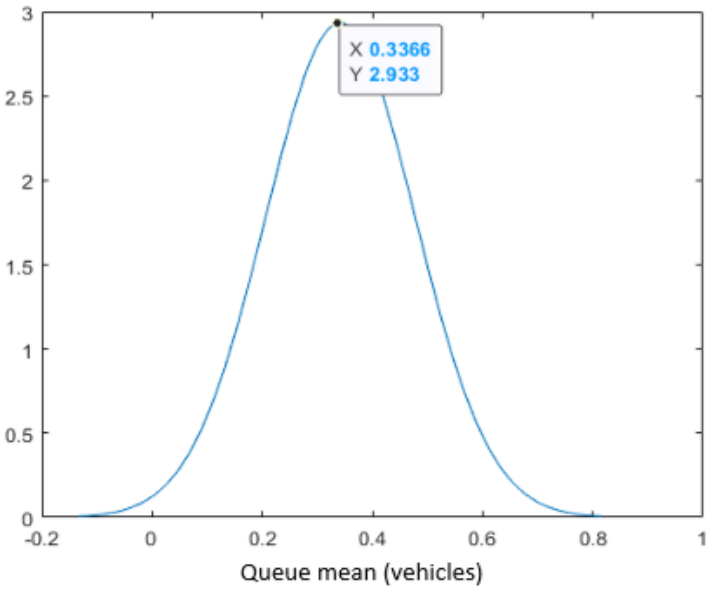


Figure 17: Normal distribution of the mean of queue length (case 2)

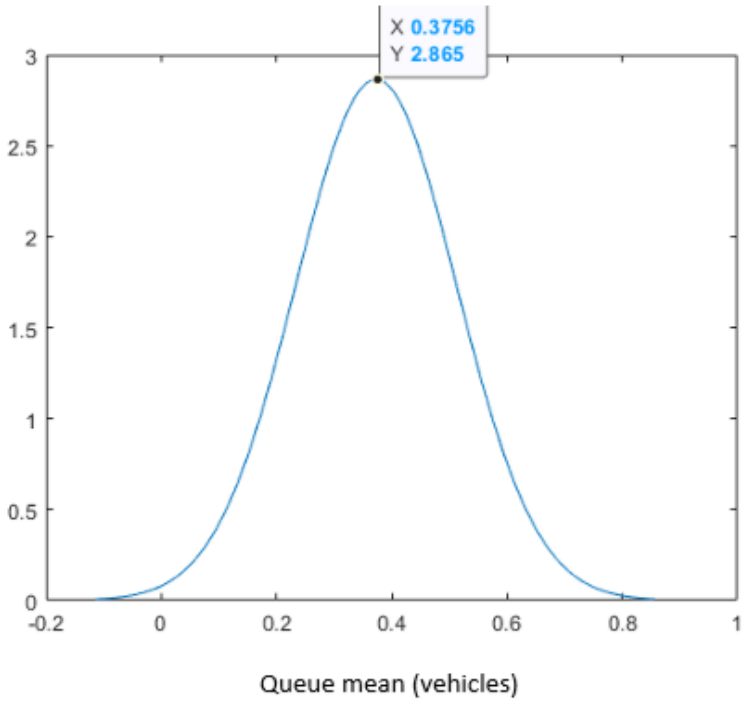


Figure 18: Normal distribution of the mean of queue length (case 3(10))

4.1.2 Unsaturated network (inflow rate=2,5)

In real road networks, the destination and origin of vehicles is not so limited. For a network to have same set of origin and destination points is hence impractical and gives rise to a saturated network. We saw in the previous section how the sluggishness sets in at later stages and the data becomes a bit muddled. So, we reduced the inflow value to 2,5 which mimics real life scenarios better. The data is condensed and mapped over each other since the terms have been introduced in the previous section.

The NFD distribution shows clearly how the peak is best reached by case 2 and how the slopes on both end are demarcated well. All the cases are shown in one graph and also as separate entities for unsaturated network for fair comparison.

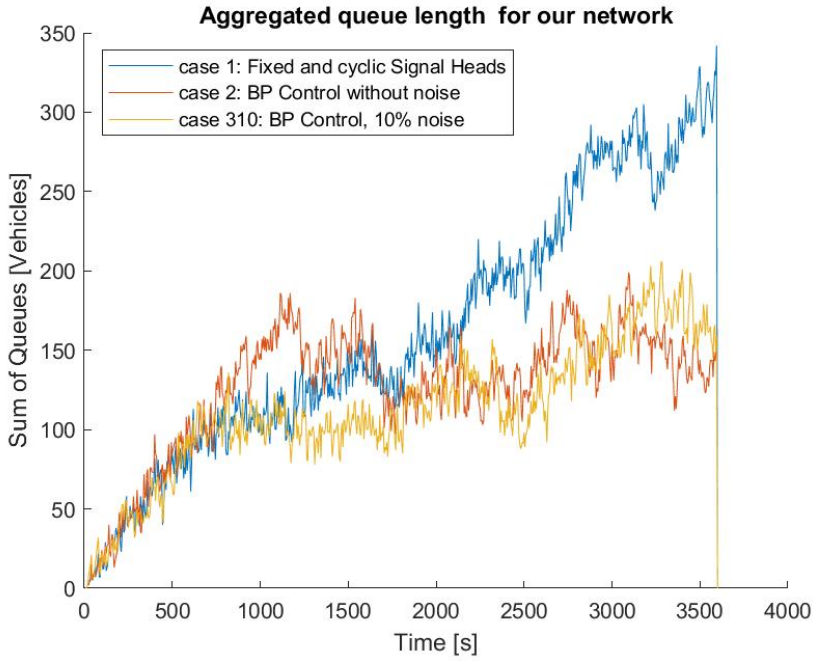


Figure 19: Aggregated queue length for our network(noise=10%)

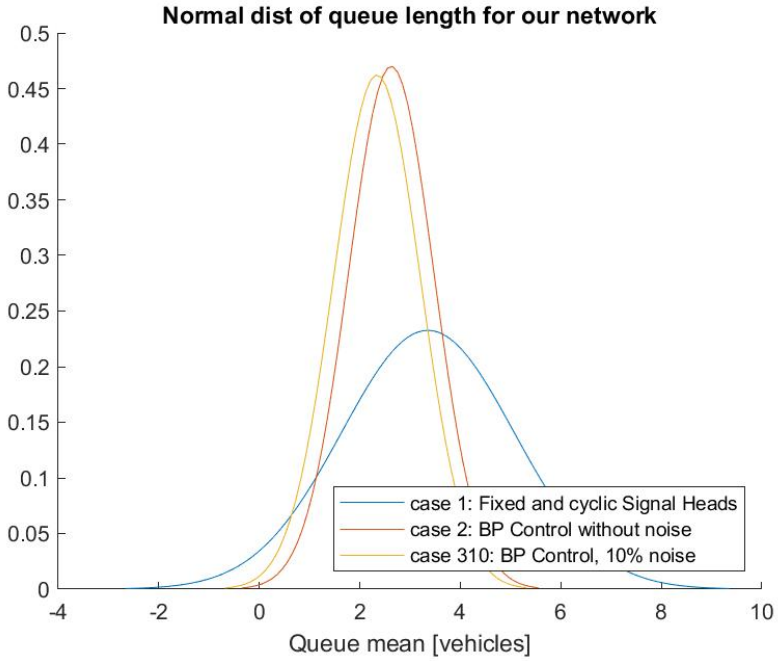


Figure 20: Normal distribution of the mean of queue length (noise=10%)

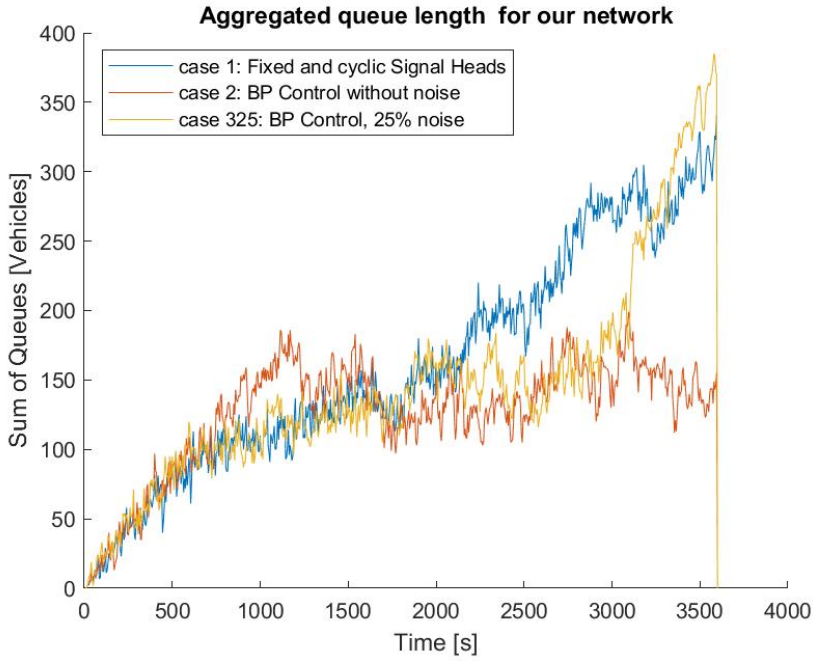


Figure 21: Aggregated queue length for our network (noise=25%)

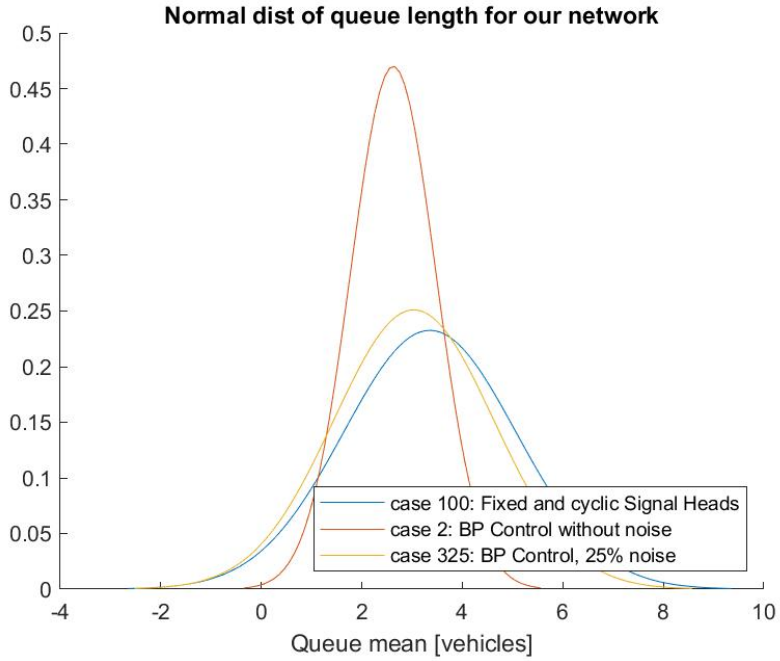


Figure 22: Normal distribution of the mean of queue length (noise=25%)

Thus, from all conclusive details of queue length and network flow diagram, we solidify over assumption of max weight algorithm's supremacy over fixed cycle. Also, we witnessed how the introduction of noise in our measurement techniques causes abrupt and sudden spikes which can lead to a longer queue estimation. This in itself is a major factor for introduction of particle filtering.

4.2 Overall results and conclusion

The four fundamental cases are computed for queue lengths (primary parameter), travel time, average speed and the number of vehicles departed (secondary parameters) from the lane selected. The results for all the cases seem to follow a common shape, which is understandable. They all follow the basic network flow diagram which implies that the queue length on an average would increase with the increase in number of vehicles, but the flow of vehicle would reach it's optimum at a certain state called 'critical density'. Before this point there is a steady increase in the flow of vehicles while after it there is congestion in traffic flow.

The best case in all the four mentioned cases has the highest value of vehicular flow (at critical density), average speed and least travel time when compared individually with all the cases' inflow rates. They are evaluated separately for two sub-cases under case 3. Thus, we have two sets of readings for the noise addition of 10% and 25% respectively.

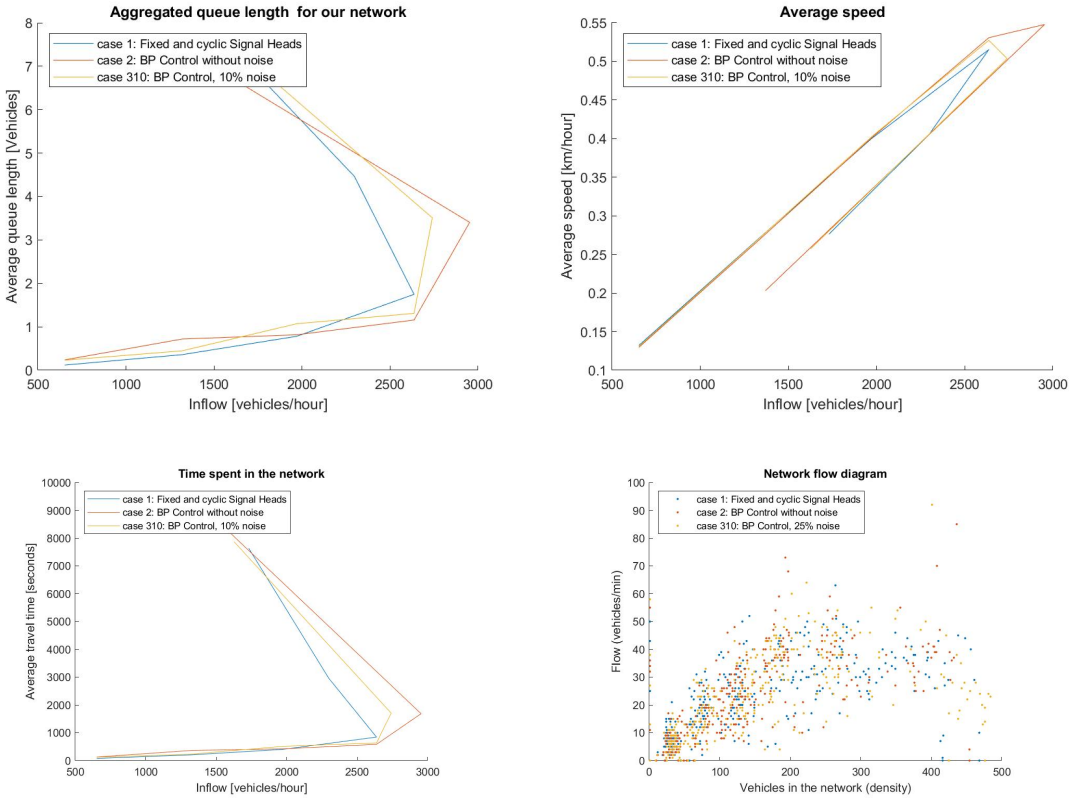


Figure 23: Overall Results. Cases: 1, 2 (MW alg.), 310 (MW alg., noise=10%) and 410 (MW alg., noise=10%, filtered). Numbers next to the points correspond to the OD matrix multiplier. Top-left: average queue length per vehicle inflow. Top-right: average speed per vehicle inflow. Bottom-left: average travel time per vehicle inflow. Bottom-right: total departed vehicles per vehicle inflow.

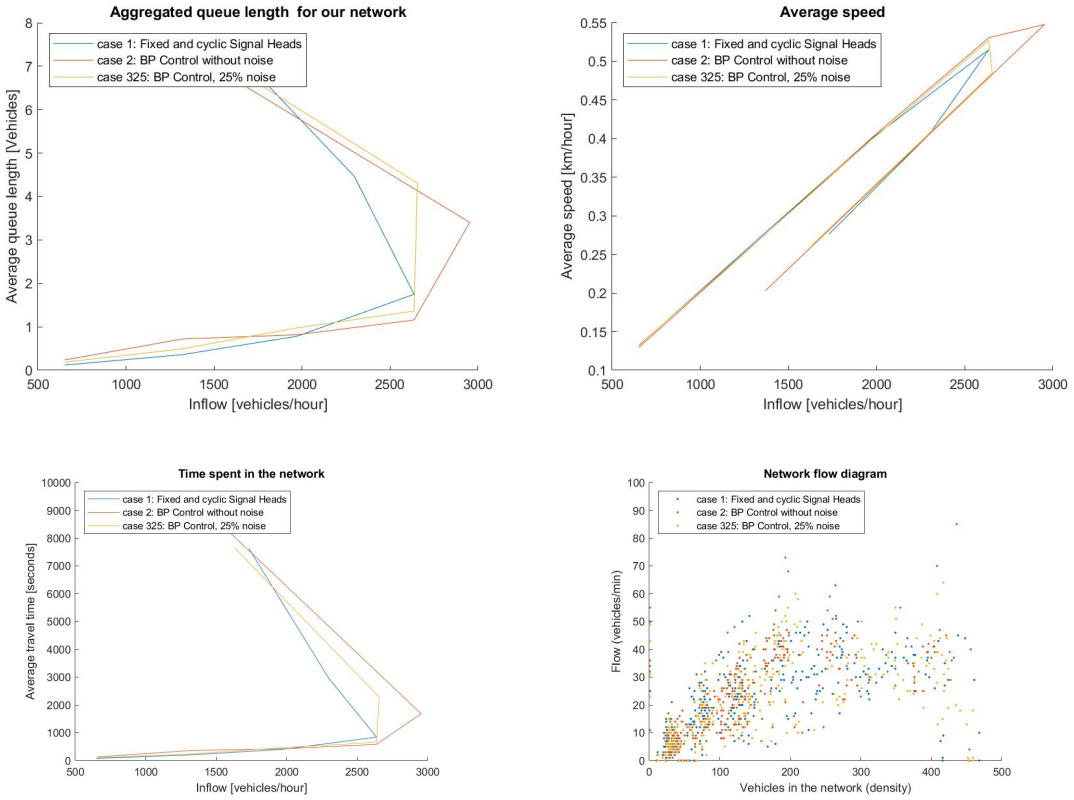


Figure 24: Overall Results. Cases: 1, 2 (MW alg.), 325 (MW alg., noise=10%) and 425 (MW alg., noise=10%, filtered). Numbers next to the points correspond to the OD matrix multiplier. Top-left: average queue length per vehicle inflow. Top-right: average speed per vehicle inflow. Bottom-left: average travel time per vehicle inflow. Bottom-right: total departed vehicles per vehicle inflow.

The results reaffirm our beliefs that the performance of the network under max-weight algorithm exceeds that of fixed-cyclic system by a fair margin. The two most important parameters, 'queue length' and 'travel time' both follow an approximate "U" shape. This is due to the saturated network condition which tends to form bottlenecks and deadlocks. Due to this, at the end of the simulation the total amount of vehicles served is substantially less.

The max-weight algorithm controllers works best in case 2, which is the ideal condition without any added noise. The cases 310 and 325 with added noise still fare better than the fixed cycle case, which implies that even with noisy measurements the algorithm should take precedence. Maximum inflow handled by the max-weight algorithm is in the ideal case (case 2) which amounts to 2900 vehicles/hr which is significantly higher than the 2500 vehicles/hr in fixed cyclic case.

The NFD looks a bit muddled, so just to see the critical density and the difference between the cases, they are shown individually below. Each NFD case contains all the inflow rate parameters and thats why the data points are a lot more.

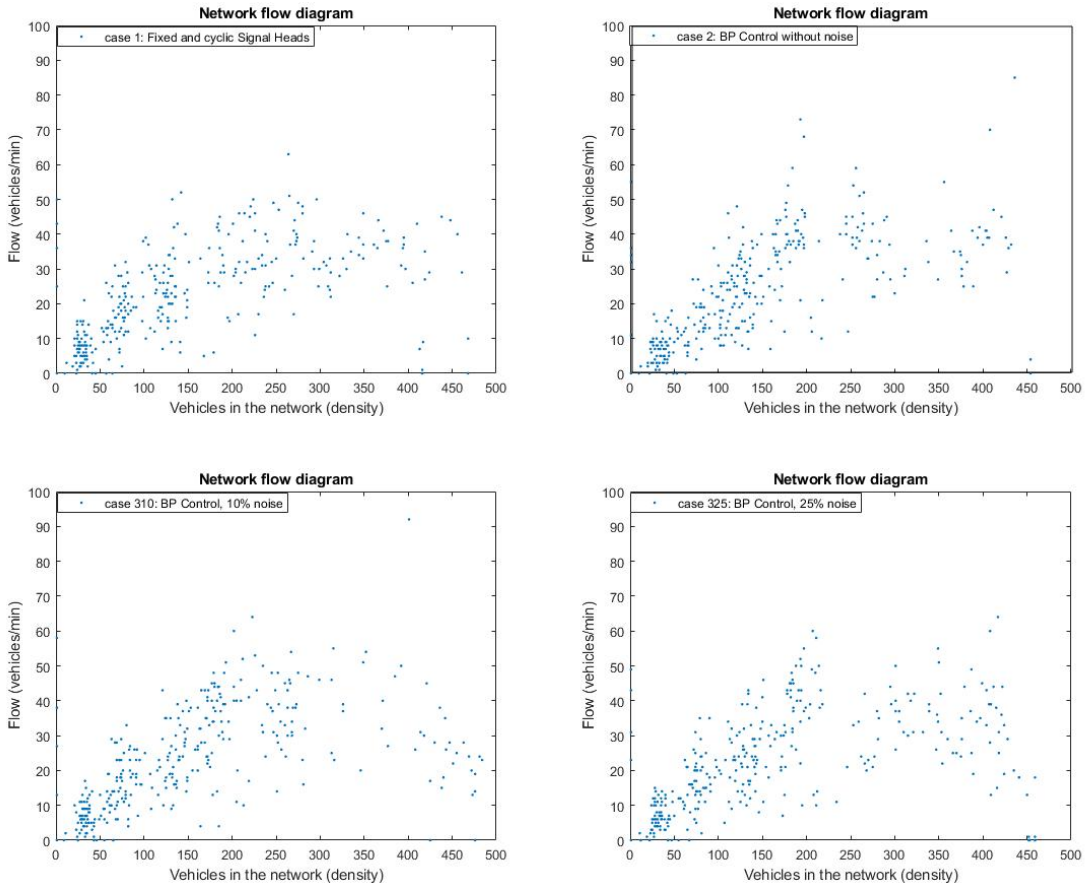


Figure 25: NFD Results for case 1, case 2, case 310 and case 325

5 Future developments

In order to simulate this in an actual urban settlement, parallel computation is must. To communicate between multiple signal controllers, they should be able to access the link-lane information about vehicles simultaneously. The use of commands like parfor, spmd and parfeval couldn't work. In VISSIM, there is a provision to get vehicle information for whole network but it's not possible to select few lanes and implement the case in parallel. Nonetheless, it does work in sequential scenarios.

Along with this, the particle filter should be applied on the network. I have provided the theory based on the references I came across and also the algorithm developed by [3] has to be tweaked the slightest as mentioned in 2. This should be the first direction to explore in terms of particle filtering for queue determination.

The two aspects mentioned above are very immediate and apparent. On a larger scale, to achieve success with the algorithm in a non-homogeneous environment would be a considerable feat. The complexity of the task would increase with size, geometry and obstructions in the most glaring way.

References

[1] PTV Group, *PTV Vissim*. Available: https://www.ptvgroup.com/en/solutions/products/ptv-vissim/

[2] R. Savolainen, *Modeling and analyzing Helsinkiâ^{TM}s traffic network using a microscopic simulator*, B.S. thesis, Aalto University, Espoo, FI, May 2018.

[3] Pallarès Ciudad, Ivan, *Modeling and analyzing Helsinkiâ^{TM}s traffic network using a microscopic simulator*, M.S. thesis, Aalto University, Espoo, FI, June 2019. Available: https://upcommons.upc.edu/bitstream/handle/2117/183593/thesis.pdf?sequence=1&isAllowed=y

[4] M. J. Park, *Three phase traffic theory*, December 2012. Available: https://guava.physics.uiuc.edu/~nigel/courses/569/Essays_Fall2012/Files/Park.pdf

[5] B. S. Kerner, *The Physics of Traffic: empirical freeway pattern features, engineering applications, and theory*. Berlin: Springer cop., 2004.

[6] Y. Sugiyama, M. Fukui, M. Kikuchi, K. Hasebe, A. Nakayama, K. Nishinari, S. Tadaki and S. Yukawa, "Traffic jams without bottlenecks-experimental evidence for the physical mechanism of the formation of a jam", *New Journal of Physics*, vol. 10, March 2008. Available: https://iopscience.iop.org/article/10.1088/1367-2630/10/3/033001/pdf

[7] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936â“1948, Dec. 1992.

[8] T. Charalambous, H. Farhadi, M. Taki, B. KulcsÃ¡r, and H. Wymeersch, "Back-pressure traffic signal control with noisy queue information".

[9] M. J. Neely. "Notes on Backpressure Routing",

textitEE 649 Stochastic Network Optimization, Spring 2011. Available: url-https://www.win.tue.nl/ sem/2MMS40/Chapter8.pdf [Accessed: June, 2019].

[10] T. Bonald, A. ProutiÃre (2003). Insensitive bandwidth sharing in data networks Queueing Systems 44 (1), 69â“100. Available: https://www.win.tue.nl/~sem/2MMS40/Chapter8.pdf [Accessed: June, 2019].

[11] A. Zaidi, B. Kulcsar, H. Wymeersch (2016), "Back Pressure Traffic Signal Control with Fixed and Adaptive Routing for Urban Vehicular Network". *IEEE transactions on intelligent transportation systems*, vol. 17(8), pp. 2134-2143. Available: http://publications.lib.chalmers.se/records/fulltext/231155/local_231155.pdf

[12] PTV Group, *PTV Vissim 11 User Manual*. PTV AG, Karlsruhe, Germany, 2018.

[13] U. E. D. City of Helsinki, "LiikennemÃÃrÃt helsingissÃ". Available: https://www.avoindata.fi/data/fi/dataset/liikennemaarat-helsingissa

[14] Google Maps, "Helsinki traffic state". [Accessed: April, 2019].

[15] Microsoft, "What Is a COM Interface?". Available: urlhttps://docs.microsoft.com/en-us/windows/win32/learnwin32/what-is-a-com-interface- [Accessed: May, 2019].

[16] T. Tettamanti, "A practical manual for Vissim COM programming in Matlab - 3rd edition for Vissim version 9 and 10", January 2018. Available: https://www.researchgate.net/publication/322232068_A_practical_manual_for_Vissim_COM_programming_in_Matlab_-_3rd_edition_for_Vissim_version_9_and_10

A OD Matrix

Figure A1 shows the Origin-Destination (OD) matrix used during the simulations. The first column and row signifies origin zone and destination zone respectively. The matrix contains information of the number of vehicles travelling from one zone to the other during 30 minutes of simulation time.

Destination matrix.PNG

12 x 12			1	2	3	4	5	6	7	8	9	10	11	12
	Name													
		Sum	55,00	55,00	55,00	55,00	55,00	55,00	55,00	55,00	55,00	55,00	55,00	55,00
1		55,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
2		55,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
3		55,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
4		55,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
5		55,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
6		55,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00	5,00
7		55,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00	5,00
8		55,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00	5,00
9		55,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00	5,00
10		55,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00	5,00
11		55,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00	5,00
12		55,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	0,00

Figure A1: Origin zone-Destination zone matrix

B Network Data Table

The main function of this table is to specify how a controller works in our network when we apply various algorithms. Each row signifies a controller at an intersection. We have a generic network with a simple link-lane correlation, hence the factor complex is null for all.

The break down of every column of a row would tell us how the vehicles would move during a phase. Each cell is a matrix of size [6*1]; as two signal heads operate on three lanes each. 'PH1INLinks' and 'PH1INLanes' when used in conjunction forms a link-lane combination, which as the name suggests contains the information of incoming link-lanes. The following two columns indicate how the vehicles move out of the link-lanes in three respective directions; straight, right and left. Similarly, the following four columns are direction indicators for phase 2.

Data Table.PNG

1 SignalController	2 Phases	3 Complex	4 Ph1INLinks	5 Ph1INLanes	6 Ph1OUTLinks	7 Ph1OUTLanes	8 Ph2INLinks	9 Ph2INLanes	10 Ph2OUTLinks	11 Ph2OUTLanes
1	2	0	[16;16;16;31;31;31]	[1;1;1;1;1;1]	[32;33;1;1;15;32]	[1;1;1;1;1;1]	[34;34;34;66;66...	[1;1;1;1;1;1]	[33;1;15;15;32;33]	[1;1;1;1;1;1]
2	2	0	[30;30;30;10;10;10]	[1;1;1;1;1;1]	[2;9;66;66;35;2]	[1;1;1;1;1;1]	[1;1;1;67;67;67]	[1;1;1;1;1;1]	[35;2;9;9;66;35]	[1;1;1;1;1;1]
3	2	0	[3;3;3;68;68;68]	[1;1;1;1;1;1]	[67;4;5;5;70;67]	[1;1;1;1;1;1]	[2;2;2;74;74;74]	[1;1;1;1;1;1]	[4;5;70;70;67;4]	[1;1;1;1;1;1]
4	2	0	[8;8;8;70;70;70]	[1;1;1;1;1;1]	[7;3;58;58;64;7]	[1;1;1;1;1;1]	[6;6;6;59;59;59]	[1;1;1;1;1;1]	[64;7;3;3;58;64]	[1;1;1;1;1;1]
5	2	0	[28;28;28;64;64;64]	[1;1;1;1;1;1]	[26;8;27;27;63;26]	[1;1;1;1;1;1]	[29;29;29;23;23]	[1;1;1;1;1;1]	[63;26;8;8;27;63]	[1;1;1;1;1;1]
6	2	0	[14;14;14;15;15;15]	[1;1;1;1;1;1]	[17;16;60;60;61;17]	[1;1;1;1;1;1]	[13;13;13;36;36...	[1;1;1;1;1;1]	[61;17;16;16;60;61]	[1;1;1;1;1;1]
7	2	0	[12;12;12;9;9;9]	[1;1;1;1;1;1]	[13;10;59;59;11;13]	[1;1;1;1;1;1]	[7;7;7;60;60;60]	[1;1;1;1;1;1]	[11;13;10;10;59;11]	[1;1;1;1;1;1]
8	2	0	[25;25;25;11;11;11]	[1;1;1;1;1;1]	[19;12;23;23;24;19]	[1;1;1;1;1;1]	[26;26;26;18;18...	[1;1;1;1;1;1]	[24;19;12;12;23;24]	[1;1;1;1;1;1]
9	2	0	[20;20;20;61;61;61]	[1;1;1;1;1;1]	[22;14;18;18;62;22]	[1;1;1;1;1;1]	[19;19;19;21;21...	[1;1;1;1;1;1]	[62;22;14;14;18;62]	[1;1;1;1;1;1]

Figure B1: Network signal controller table for our network

C MATLAB Code

C.1 Import network data from .csv file function

The data is first transferred from VISSIM to a '.csv, file and then passed onto all three networks. This data contains the information of links and lanes along with the network data table.

```
file = 'Network_data_table_II.csv';
%file = 'Network_data_table.csv';
NSISC = readtable(file);

for row=1:size(NSISC,1)
    for column=4:(NSISC{row,2}*4+3)
        temp = NSISC{row,column}{:};
        temp = strrep(temp,'[','');
        temp = strrep(temp,']','');
        temp  = strsplit(temp,';');
        matrix=[];
        for i=1:size(temp,2)
            matrix = [ matrix ; str2double(temp(i)) ];
        end
        NSISC{row,column}{:} = matrix;
    end
end

% Table of Links and Lanes used for the SC (Where the queues
    take place)
NSIL=table();
test_table=table();
for row=1:size(NSISC,1)
    for column=1:(NSISC{row,2}*2)
        for i=1:size(NSISC{row,((column-1)*2+4)}{:},1)
            for j=1:size(NSISC{row,((column-1)*2+5)}{:},2)
                test_table.Link = NSISC{row,((column-1)*2+4)
                    }{:}(i,j);
                test_table.Lane = NSISC{row,((column-1)*2+5)
                    }{:}(i,j);
                NSIL=[NSIL;test_table];
            end
        end
    end
end

NSIL=unique(NSIL);
%NSIL(1,:)=[];
```

```
36
37 % save 'nine_network_sc' NSISC
38 % save 'nine_lanes_table_sc' NSIL
39 save 'nine_network_sc_II' NSISC
40 save 'nine_lanes_table_sc_II' NSIL
41 output_data1 = NSISC
42 output_data2 = NSIL
```

C.2 Simulation scripts

These scripts process our parameters and store it in a '.mat'file.

```
1 %Fixed cycle algorithm
2
3 Nnw2_case=1
4 % Create/Verify folders
5 if exist('results/Nnw2/case_1','dir')==0
6     mkdir('results/Nnw2/case_1');
7 end
8
9 inflow_values=[0.5,1,1.5,2,2.5,3];
10
11 l=0;
12 Q_mean=size(1,size(inflow_values,2));
13 vehperhour=size(1,size(inflow_values,2));
14 Avg_Speed_mean=size(1,size(inflow_values,2));
15 TmInNetTot_mean=size(1,size(inflow_values,2));
16
17 TSI_vehdeparted=size(1,size(inflow_values,2));
18 TSI_Number=size(1,size(inflow_values,2));
19
20 %count_arr and count_net are used for NFD representation.
     They are cells of
21 %the size of inflow values introduced in the network. Each
     cell contains a
22 %[1*60] array. The two values signify vehicle density at
     that instant
23 %(count_net) and the vehicles that have arrived at their
     destination zone
24 %(count_arr).
25
26 count_net=cell(1,size(inflow_values,2));
27 count_arr=cell(1,size(inflow_values,2));
28
29 for inflow=inflow_values
```

```
30 l=l+1;
31 % SIMULATION DATA
32
33 Tsig=20;                              % Control Period Time
34 seconds=3600;                         % 3600 (simulation time in
       seconds)
35 %seconds=120;
36 sampling_time=5;
37
38 % INITIALIZATION
39
40 load('nine_network_sc_II.mat')      % Load SC Network
       Properties
41 load('nine_lanes_table_sc_II.mat') % Load SC Network
       Properties
42
43 vissim_com=actxserver('Vissim.Vissim.1100');
44 vissim_com.LoadNet([pwd '\Nnw2.inpx']);
45
46 sim=vissim_com.Simulation;
47 vnet=vissim_com.Net;
48 scs=vnet.SignalControllers;
49 links=vnet.Links;
50 demand_matrix1=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(1).Matrix;
51 demand_matrix2=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(2).Matrix;
52
53 TSI_max_phases=max(NSISC.Phases);
54
55 ind=ones(1,size(NSISC,1));
56 prev_ind=ones(1,size(NSISC,1));
57
58 Volume=zeros(size(NSIL,1),seconds/sampling_time);
59 Speed=zeros(size(NSIL,1),seconds/sampling_time);
60 Avg_Speed=zeros(size(NSIL,1),seconds/sampling_time);
61 DelayTm=zeros(size(NSIL,1),seconds/sampling_time);
62 TmInNetTot=zeros(size(NSIL,1),seconds/sampling_time);
63 Q=zeros(size(NSIL,1),seconds/sampling_time);
64 QN=zeros(size(NSIL,1),seconds/sampling_time);
65 NL=zeros(size(NSIL,1),seconds/sampling_time);
66
67 Number=strings(size(NSIL,1),seconds/sampling_time);
68 N="";
69 DZone=cell(size(NSIL,1),seconds/sampling_time);
```

```
70 % OZone=cell(size(NSIL,1),seconds/sampling_time);
71
72 vehdeparted=zeros(1,seconds);
73 veharrived=zeros(1,seconds);
                                                    % Sum of
      arrived vehs in network (per seconds or step)
74 vehevolution=zeros(1,seconds);
75
76 LinkID_list=NSIL{:,1}';
77 LaneID_list=NSIL{:,2}';
78
79 conversion_list=zeros(size(NSIL,1),4);
80 for i=1:size(NSIL,1)
81     conversion_list(LinkID_list(i),LaneID_list(i))=i;
82 end
83
84 b=NSISC{:,1}
85
86 Link_length=cell(1,size(NSIL,1));
87 Link_name=cell(1,size(NSIL,1));
88 for i=1:size(NSIL,1)
89     Link_length{i}=links.ItemByKey(LinkID_list(i)).AttValue(
         'Length2D');%Link length in m
90     Link_name{i}=links.ItemByKey(LinkID_list(i)).AttValue('
         Name');%Link length in m
91 end
92
93 for row=1:demand_matrix1.RowCount()
94     for col=1:demand_matrix1.ColCount()
95             matrix_cell=demand_matrix1.GetValue(row,col)*
                 inflow;
96             demand_matrix1.SetValue(row,col,matrix_cell)
97     end
98 end
99
100 for row=1:demand_matrix2.RowCount()
101     for col=1:demand_matrix2.ColCount()
102         matrix_cell=demand_matrix2.GetValue(row,col)*inflow;
103         demand_matrix2.SetValue(row,col,matrix_cell)
104     end
105 end
106 %% SIMULATION
107 sim.set('AttValue','SimRes',1);
                                          % Resolution %Vivek
108 eval=vissim_com.Evaluation;
```

```
109 set(vissim_com.Graphics,'AttValue','Quickmode',1);
110
111
112 for k=1:seconds
113     vissim_com.Simulation.RunSingleStep;                    % Run next Step (1 second)
114     vehdeparted(k)=vnet.Vehicles.GetDeparted.Count;
115     vehevolution(k)=vnet.Vehicles.count;
116     if k>1
117         veharrived(k)=vehevolution(k-1)-vehevolution(k);    % Vehicles departed per step (per second)
118     end
119     %-------------------------------------------------------------------------

120     % Initialize Signal Controllers per k==1s
121     if k==1
122         message=strcat('Progress: ',num2str(k/seconds*100,4),'% [',num2str(k),'s]');
123         disp(message)
124
125
126     end
127     %-------------------------------------------------------------------------

128
129     if mod(k,sampling_time)==0
130
131         %use the code below to call out all the selected lanes from the table
132         for i=1:size(NSIL,1)
133             Vehicle_Attributes = cell2mat(links.ItemByKey(NSIL{i,1}).Lanes.ItemByKey(NSIL{i,2}).Vehs.GetAll);
134
135             if(size(Vehicle_Attributes,1)>0)
136                 DZone{i,k/sampling_time}=size(1,size(Vehicle_Attributes,1));
137 %                 OZone{i,k/sampling_time}=size(1,size(Vehicle_Attributes{i,k/sampling_time},1));
138
139                 for z=1:size(Vehicle_Attributes,1)
140                     Volume(i,k/sampling_time)= Volume(i,k/sampling_time)+1;
```

```
141
142                         dtm=Vehicle_Attributes(z).AttValue('DelayTm');
143                         tin=Vehicle_Attributes(z).AttValue('TmInNetTot');
144                         spd=Vehicle_Attributes(z).AttValue('Speed');
145                         no=num2str(Vehicle_Attributes(z).AttValue('No'));
146
147                         Number(i,k/sampling_time)=append(no,'-',Number(i,k/sampling_time));
148
149                         %This is the unique identification number of the
150                         %vehicle entering the lane. In order to avoid multiple
151                         %instances of same vehicle getting counted in Ozone
152                         %that measures origin of vehicle whose values are being
153                         %computed.
154                         %fprintf("no-dest");
155                         DZone{i,k/sampling_time}(z)=str2double(Vehicle_Attributes(z).AttValue('DestZone'));
156 %                         OZone{i,k/sampling_time}(z)=str2double(Vehicle_Attributes{i,k/sampling_time}(z).AttValue('OrigZone'));
157                         %Destination Zone and origin zone are the designated zones of vehicles
158                         %present in the lanes during a particular simulation
159                         %step. These variables helps us view where the traffic
160                         %flow is from. Helpful if measuring other smaller parts
161                         %of the network and we want a part of OD matrix rather
162                         %than the whole set up. In the results section, the
163                         %counts of vehicles and zones are listed for each step.
164                         %In our case of two signal taken out of the network,
```

```
165                         %each and every zone is in use.
166
167                         Speed(i,k/sampling_time)=spd+Speed(i,k/
                                sampling_time);
168                         if spd<4
169                             Q(i,k/sampling_time)=Q(i,k/
                                    sampling_time)+1;
170                         end
171                         DelayTm(i,k/sampling_time)=DelayTm(i,k/
                                sampling_time)+dtm;
172                         TmInNetTot(i,k/sampling_time)=TmInNetTot
                                (i,k/sampling_time)+tin;
173                     end
174                 end
175                 if(Speed(i,k/sampling_time)~=0)
176                     Avg_Speed(i,k/sampling_time)=Speed(i,k/
                            sampling_time)/size(NSIL,1);
177                 end
178             end
179         end
180 end
181
182 %The unique IDs of vehicles in the network is calculated
        below. Further
183 %stored in TSINumber.
184 Number;
185 Num_tmp="";
186 for i=1:size(NSIL,1)
187     for j=1:seconds/sampling_time
188         Num_tmp=append(Num_tmp,Number(i,j));
189     end
190 end
191 Num_tmp;
192 Num_tmp_2=split(Num_tmp,"-");
193 Num_tmp_3=str2double(unique(Num_tmp_2));
194 Num_tmp_3=Num_tmp_3(~isnan(Num_tmp_3));
195
196 Avg_Speed_mean(l)=mean(sum(Avg_Speed))/size(Avg_Speed,1);
197 TmInNetTot_mean(l)=mean(sum(TmInNetTot)/size(TmInNetTot,1));
198
199 Q_temp=zeros(size(Q,1),seconds/sampling_time);
200 Q_sum=zeros(1,seconds/sampling_time);
201 for i=1:seconds/sampling_time
202     for j=1:size(Q,1)
203         Q_temp(j,i)=Q(j,i);
```

```
204         end
205         Q_sum(i)=sum(Q_temp(:,i));
206
207 end
208
209 %We will get a plot of the sum of queues over a period of
        one hour, inflow
210 %set to the last inflow value.
211 plot(sampling_time:sampling_time:seconds,Q_sum);
212
213 Q_avg=sum(Q_temp)/size(Q,1);
214 Q_mean(l)=mean(Q_avg);
215 Q_avg_max=mean(Q_avg)+std(Q_avg)*3.5;
216 Q_avg_min=mean(Q_avg)-std(Q_avg)*3.5;
217 interval=0.1;
218
219 x=[Q_avg_min:interval:Q_avg_max];
220 y=normpdf(x,mean(Q_avg),std(Q_avg));
221
222 %Parameters measured over every inflow value as an array
        element.
223 vehperhour(l)=sum(vehdeparted);
224 %vehperhour is the count of departed vehicles from the
        previously marked
225 %parking space during each simulation.
226 TSI_Number(l)=size(Num_tmp_3,1);
227 %TSI_Number is the sum of unique vehicles present during
        each inflow rate.
228
229
230 %
    ------------------------------------------------------------------------

231 %Aggregated sum of queues and their mean's normal
        distribution-inflow 2,5
232 if(inflow_values==2.5)
233         C1Q_sum_25=Q_sum;
234         C1x_25=x;
235         C1y_25=y;
236 end
237
238
239 %
    ------------------------------------------------------------------------
```

```
240 %NFD parameters for all the inflow values are grouped
    together here and
241 %stored as array values. Hence, for our calculation we
    extract all inflow
242 %values based fundamental diagram.
243
244 %Number;
245 temp="";
246 List=strings(1,seconds/sampling_time);
247 for i=1:seconds/sampling_time
248     for j=1:size(NSIL,1)
249         temp=append(temp,Number(j,i));
250     end
251     List(1,i)=temp;
252     temp="";
253 end
254 %List;
255 List2=cell(1,seconds/sampling_time);
256 sz=zeros(1,seconds/sampling_time);
257 for i=1:seconds/sampling_time
258     %With the help of this expression, we get the ID of
        vehicles involved
259     %every 5 mins.
260     List2{i}=sort(str2double(split(List{i},"-")));
261     sz(i)=size(List2{i},1);
262 end
263 %List2;
264 %sz;
265
266 %count tells us the number of vehicles that have left from
    the network or
267 %in other words have reached their destination.
268 k=[];
269 ID={};
270 for i=1:seconds/sampling_time
271     for m=1:i
272       k=[k,List2{m}'];
273     end
274
275     n=rmmissing(sort(k(:)'));
276     g=rmmissing(unique(k));
277     t=histc(n,g);
278
279     for o=1:size(t,2)
280         ID{i,o}=[g(o),t(o)];
```

```
281         end
282
283         k=[];
284     end
285
286     idx=[];
287
288     Last_ID={};
289     for  i=1:size(ID,2)
290         Last_ID{i}=ID{size(ID,1),i};
291     end
292
293     count=1;
294     moments=zeros(1,size(ID,2));
295     for  i=1:size(ID,2)
296         for  j=1:size(ID,1)
297             if(~isempty(ID{j,i}))
298                 if(ID{j,i}(2)==Last_ID{1,i}(2))
299                     moments(count)=j+1;
300                     count=count+1;
301                     break;
302                 end
303             end
304         end
305     end
306
307     count_arr{l}=zeros(1,60);
308     time_stamp=[1:12:size(ID,1)];
309     for  i=1:size(moments,2)
310         for  j=1:size(time_stamp,2)
311             if(moments(1,i)<time_stamp(1,j))
312                 count_arr{l}(1,j)=count_arr{l}(1,j)+1;
313                 break;
314             end
315         end
316     end
317
318     count_net{l}=zeros(1,60);
319     for  i=1:size(sz,2)
320         if(mod(i,12)==0)
321             i;
322             count_net{l}(i/12)=sz(i);
323         end
324     end
325     end
```

```
326 C1Q_mean=Q_mean;
327 C1TSI_Number=TSI_Number;
328 C1vehperhour=vehperhour;
329 C1Avg_Speed_mean=Avg_Speed_mean;
330 C1TmInNetTot_mean=TmInNetTot_mean;
331
332 C1NFD_count_net=count_net;
333 C1NFD_count_arr=count_arr;
334
335 result=strcat('results\Nnw2\case_1\case_1','.mat');
336 save(result,'NSIL','NSISC','C1Q_mean', 'C1vehperhour', '
    C1Avg_Speed_mean', 'C1TmInNetTot_mean','C1TSI_Number','
    C1y_25','C1x_25','C1Q_sum_25','C2NFD_count_net','
    C2NFD_count_arr');
```

```
1 %Maximum-weight algorithm without noise addition
2
3 Nnw2_case=2
4 % Create/Verify folders
5 if exist('results/Nnw2/case_2','dir')==0
6     mkdir('results/Nnw2/case_2');
7 end
8
9 inflow_values=[0.5,1,1.5,2,2.5,3];
10
11 l=0;
12
13 Q_mean=size(1,size(inflow_values,2));
14 vehperhour=size(1,size(inflow_values,2));
15 Avg_Speed_mean=size(1,size(inflow_values,2));
16 TmInNetTot_mean=size(1,size(inflow_values,2));
17
18 TSI_Number=size(1,size(inflow_values,2));
19
20 %count_arr and count_net are used for NFD representation.
    They are cells of
21 %the size of inflow values introduced in the network. Each
    cell contains a
22 %[1*60] array. The two values signify vehicle density at
    that instant
23 %(count_net) and the vehicles that have arrived at their
    destination zone
24 %(count_arr).
25
26 count_net=cell(1,size(inflow_values,2));
```

```
27 count_arr=cell(1,size(inflow_values,2));
28
29 for inflow=inflow_values
30 l=l+1;
31 % SIMULATION DATA
32
33 Tsig=20;                              % Control Period Time
34 seconds=3600;                         % 3600 (simulation time in
       seconds)
35 sampling_time=5;
36
37 % INITIALIZATION
38
39 load('nine_network_sc_II.mat')      % Load SC Network
       Properties
40 load('nine_lanes_table_sc_II.mat') % Load SC Network
       Properties
41
42 vissim_com=actxserver('Vissim.Vissim.1100');
43 vissim_com.LoadNet([pwd '\Nnw2.inpx']);
44
45 sim=vissim_com.Simulation;
46 vnet=vissim_com.Net;
47 scs=vnet.SignalControllers;
48 links=vnet.Links;
49 demand_matrix1=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(1).Matrix;
50 demand_matrix2=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(2).Matrix;
51
52 TSI_max_phases=max(NSISC.Phases);
53
54 ind=ones(1,size(NSISC,1));
55 prev_ind=ones(1,size(NSISC,1));
56
57 Volume=zeros(size(NSIL,1),seconds/sampling_time);
58 Speed=zeros(size(NSIL,1),seconds/sampling_time);
59 Avg_Speed=zeros(size(NSIL,1),seconds/sampling_time);
60 DelayTm=zeros(size(NSIL,1),seconds/sampling_time);
61 TmInNetTot=zeros(size(NSIL,1),seconds/sampling_time);
62 Q=zeros(size(NSIL,1),seconds/sampling_time);
63 QN=zeros(size(NSIL,1),seconds/sampling_time);
64 NL=zeros(size(NSIL,1),seconds/sampling_time);
65
66 Number=strings(size(NSIL,1),seconds/sampling_time);
```

```
67 N="";
68 DZone=cell(size(NSIL,1),seconds/sampling_time);
69 OZone=cell(size(NSIL,1),seconds/sampling_time);
70
71 vehdeparted=zeros(1,seconds);
72
73 LinkID_list=NSIL{:,1}';
74 LaneID_list=NSIL{:,2}';
75
76 conversion_list=zeros(size(NSIL,1),max(NSISC.Phases));
77 for i=1:size(NSIL,1)
78     conversion_list(LinkID_list(i),LaneID_list(i))=i;
79 end
80
81
82 b=NSISC{:,1};
83 shs=cell(size(NSISC,1),TSI_max_phases);
84 for i=1:size((NSISC{:,1}),1)
85     for j= 1:NSISC.Phases(i)
86         shs{i,j}=scs.ItemByKey(b(i)).SGs.GetAll{j};
87     end
88 end
89
90 % Record of State of Signal Heads
91 Signal_heads_states=ones(size(NSISC,1),TSI_max_phases,
    seconds);
92 for k=1:seconds
93  for j=1:size(NSISC,1)
94    if NSISC{j,2}~=TSI_max_phases
95      for i=NSISC{j,2}+1:TSI_max_phases
96          Signal_heads_states(j,i,k)=0;
97       end
98     end
99  end
100 end
101
102 Link_length=cell(1,size(NSIL,1));
103 Link_name=cell(1,size(NSIL,1));
104 for i=1:size(NSIL,1)
105     Link_length{i}=links.ItemByKey(LinkID_list(i)).AttValue(
        'Length2D');%Link length in m
106     Link_name{i}=links.ItemByKey(LinkID_list(i)).AttValue('
        Name');%Link length in m
107 end
108
```

```
109 for row=1:demand_matrix1.RowCount()
110     for col=1:demand_matrix1.ColCount()
111             matrix_cell=demand_matrix1.GetValue(row,col)*
                    inflow;
112             demand_matrix1.SetValue(row,col,matrix_cell)
113     end
114 end
115
116 for row=1:demand_matrix2.RowCount()
117     for col=1:demand_matrix2.ColCount()
118         matrix_cell=demand_matrix2.GetValue(row,col)*inflow;
119         demand_matrix2.SetValue(row,col,matrix_cell)
120     end
121 end
122
123
124 %% SIMULATION
125 sim.set('AttValue','SimRes',1);
                                        % Resolution %Vivek
126 eval=vissim_com.Evaluation;
127 set(vissim_com.Graphics,'AttValue','Quickmode',1);
128
129
130 for k=1:seconds
131     vissim_com.Simulation.RunSingleStep;                    %
        Run next Step (1 second)
132     vehdeparted(k)=vnet.Vehicles.GetDeparted.Count;
133     %
        ------------------------------------------------------------------------

134     % Initialize Signal Controllers per k==1s
135     if k==1
136         message=strcat('Progress: ',num2str(k/seconds*100,4)
                ,'% [',num2str(k),'s]');
137         disp(message)
138             for j=1:size(NSISC,1)
139                 for i=1:NSISC{j,2}
140                     set(scs.ItemByKey(b(j)).SGs.ItemByKey(i)
                            ,'AttValue','ContrByCOM',1);
141                     %The number 11 is the controller number.
                            Would change with the selection of
                        intersection.
142                     set(scs.ItemByKey(b(j)).SGs.ItemByKey(i)
                            ,'AttValue','State',1);
143                     if i==1
```

```
144                         set(scs.ItemByKey(b(j)).SGs.
                                ItemByKey(i),'AttValue','State'
                                ,3);
145                     end
146                 end
147             end
148     end
149     %
```

```
150
151     if mod(k,sampling_time)==0
152
153         %use the code below to call out all the selected
               lanes from the table
154         for i=1:size(NSIL,1)
155             Vehicle_Attributes = cell2mat(links.ItemByKey(
                   NSIL{i,1}).Lanes.ItemByKey(NSIL{i,2}).Vehs.
                   GetAll);
156
157             if(size(Vehicle_Attributes,1)>0)
158
159                 for z=1:size(Vehicle_Attributes,1)
160                     Volume(i,k/sampling_time)= Volume(i,k/
                           sampling_time)+1;
161
162                     dtm=Vehicle_Attributes(z).AttValue('
                           DelayTm');
163                     tin=Vehicle_Attributes(z).AttValue('
                           TmInNetTot');
164                     spd=Vehicle_Attributes(z).AttValue('
                           Speed');
165                     no=num2str(Vehicle_Attributes(z).
                           AttValue('No'));
166
167                     Number(i,k/sampling_time)=append(no,'-',
                           Number(i,k/sampling_time));
168
169                     Speed(i,k/sampling_time)=spd+Speed(i,k/
                           sampling_time);
170                     if spd<4
171                         Q(i,k/sampling_time)=Q(i,k/
                               sampling_time)+1;
172                     end
173                     DelayTm(i,k/sampling_time)=DelayTm(i,k/
```

```
                    sampling_time)+dtm;
174                 TmInNetTot(i,k/sampling_time)=TmInNetTot
                        (i,k/sampling_time)+tin;
175             end
176         end
177         if(Speed(i,k/sampling_time)~=0)
178             Avg_Speed(i,k/sampling_time)=Speed(i,k/
                    sampling_time)/size(NSIL,1);
179         end
180     end
181
182 end
183
184 %
```

```
185 % BACK PRESSURE Algorithm
186
187 if mod(k,Tsig)==0 && k<seconds
188
189     for j=1:size(NSISC,1)
190
191         % Pressure release calculation
192
193         pa=zeros(1,NSISC{j,2});
194         for phase_sc=1:NSISC{j,2}
195
196             for option_sc=1:size(NSISC{j,((phase_sc-1)
                    *4+4)}{:},1)
197                 queue_cars_sc_in=0;
198                 queue_cars_sc_out=0;
199                 for option_sc_2=1:size(NSISC{j,((
                        phase_sc-1)*4+4)}{:},2)    % IN Links
                        and Lanes + queue
200
201                     link_sc_in = NSISC{j,((phase_sc-1)
                            *4+4)}{:}(option_sc,option_sc_2);
202                     lane_sc_in = NSISC{j,((phase_sc-1)
                            *4+5)}{:}(option_sc,option_sc_2);
203
204                     if link_sc_in ~= 0 || lane_sc_in ~=
                            0
205                         queue_cars_sc_in =
                                queue_cars_sc_in + Q(
                                conversion_list(link_sc_in,
```

```
                                lane_sc_in),k/sampling_time);
206                         end
207                     end
208
209                     for option_sc_2=1:size(NSISC{j,((
                            phase_sc-1)*4+6)}{:},2)    % OUT Links
                            and Lanes + queue
210
211                         link_sc_out = NSISC{j,((phase_sc-1)
                                *4+6)}{:}(option_sc,option_sc_2);
212                         lane_sc_out = NSISC{j,((phase_sc-1)
                                *4+7)}{:}(option_sc,option_sc_2);
213
214                         if link_sc_out ~= 0 || lane_sc_out
                                ~= 0
215                             queue_cars_sc_out =
                                    queue_cars_sc_out + Q(
                                    conversion_list(link_sc_out,
                                    lane_sc_out),k/sampling_time)
                                    ;
216                         end
217                     end
218
219                     pa(phase_sc) = pa(phase_sc) + max(
                            queue_cars_sc_in - queue_cars_sc_out
                            , 0 );  % PA calc
220
221                 end
222             end
223
224             [maxx,ind(j)]=max(pa);
225             s=find(maxx-pa==0);
226             if length(s)>1
227                 ind(j)=s(randi([1 length(s)],1));
228             end
229
230             % modify the signal heads
231             if ind(j)~=prev_ind(j)
232                 set(shs{j,ind(j)},'AttValue','State',3);
                                % Green Light
233                 set(shs{j,prev_ind(j)},'AttValue','State',1)
                        ;       % Red Light
234                 prev_ind(j)=ind(j);
235             end
236
```

```
237             end         %end of junctions
238
239
240         end
241
242         for j=1:size(ind,2)
243             Signal_heads_states(j,ind(j),k)=3;
244         end
245     end
246
247     %The unique IDs of vehicles in the network is calculated
        below. Further
248     %stored in TSINumber.
249     Number;
250     Num_tmp="";
251     for i=1:size(NSIL,1)
252         for j=1:seconds/sampling_time
253             Num_tmp=append(Num_tmp,Number(i,j));
254         end
255     end
256     Num_tmp;
257     Num_tmp_2=split(Num_tmp,"-");
258     Num_tmp_3=str2double(unique(Num_tmp_2));
259     Num_tmp_3=Num_tmp_3(~isnan(Num_tmp_3));
260
261     Avg_Speed_mean(l)=mean(sum(Avg_Speed))/size(Avg_Speed,1);
262     TmInNetTot_mean(l)=mean(sum(TmInNetTot)/size(TmInNetTot,1));
263
264     Q_temp=zeros(size(Q,1),seconds/sampling_time);
265     Q_sum=zeros(1,seconds/sampling_time);
266     for i=1:seconds/sampling_time
267         for j=1:size(Q,1)
268             Q_temp(j,i)=Q(j,i);
269         end
270         Q_sum(i)=sum(Q_temp(:,i));
271
272     end
273
274     %We will get a plot of the sum of queues over a period of
        one hour, inflow
275     %set to the last inflow value.
276     plot(sampling_time:sampling_time:seconds,Q_sum)
277
278     Q_avg=sum(Q_temp)/size(Q,1);
279     Q_mean(l)=mean(Q_avg);
```

```
Q_avg_max=mean(Q_avg)+std(Q_avg)*3.5;
Q_avg_min=mean(Q_avg)-std(Q_avg)*3.5;

interval=0.1;
x=[Q_avg_min:interval:Q_avg_max];
y=normpdf(x,mean(Q_avg),std(Q_avg));

%Parameters measured over every inflow value as an array
    element.
vehperhour(l)=sum(vehdeparted);
%vehperhour is the count of departed vehicles from the
    previously marked
%parking space during each simulation.
TSI_Number(l)=size(Num_tmp_3,1);
%TSI_Number is the sum of unique vehicles present during
    each inflow rate.

%
    ------------------------------------------------------------------------

%Aggregated sum of queues and their mean's normal
    distribution-inflow 2,5

if(inflow_values==2.5)
      C2Q_sum_25=Q_sum;
      C2x_25=x;
      C2y_25=y;
end

%
    ------------------------------------------------------------------------

%NFD parameters for all the inflow values are grouped
    together here and
%stored as array values. Hence, for our calculation we
    extract all inflow
%values based fundamental diagram.

%Number;
temp="";
List=strings(1,seconds/sampling_time);
for i=1:seconds/sampling_time
     for j=1:size(NSIL,1)
          temp=append(temp,Number(j,i));
     end
```

```
315         List(1,i)=temp;
316         temp="";
317     end
318
319     List2=cell(1,seconds/sampling_time);
320     sz=zeros(1,seconds/sampling_time);
321     for i=1:seconds/sampling_time
322         %With the help of this expression, we get the ID of
                vehicles involved
323         %every 5 mins.
324         List2{i}=sort(str2double(split(List{i},"-")));
325         sz(i)=size(List2{i},1);
326     end
327
328     %count tells us the number of vehicles that have left from
            the network or
329     %in other words have reached their destination.
330     k=[];
331     ID={};
332     for i=1:seconds/sampling_time
333         for m=1:i
334           k=[k,List2{m}'];
335         end
336         n=rmmissing(sort(k(:)'));
337         g=rmmissing(unique(k));
338         t=histc(n,g);
339         for o=1:size(t,2)
340             ID{i,o}=[g(o),t(o)];
341         end
342         k=[];
343     end
344
345     idx=[];
346     Last_ID={};
347     for i=1:size(ID,2)
348         Last_ID{i}=ID{size(ID,1),i};
349     end
350
351     count=1;
352     moments=zeros(1,size(ID,2));
353     for i=1:size(ID,2)
354         for j=1:size(ID,1)
355             if(~isempty(ID{j,i}))
356                 if(ID{j,i}(2)==Last_ID{1,i}(2))
357                     moments(count)=j+1;
```

```
                    count=count+1;
                    break;
                end
            end
        end
    end

    count_arr{l}=zeros(1,60);
    time_stamp=[1:12:size(ID,1)];
    for i=1:size(moments,2)
        for j=1:size(time_stamp,2)
            if(moments(1,i)<time_stamp(1,j))
                count_arr{l}(1,j)=count_arr{l}(1,j)+1;
                break;
            end
        end
    end

    count_net{l}=zeros(1,60);
    for i=1:size(sz,2)
        if(mod(i,12)==0)
            i;
            count_net{l}(i/12)=sz(i);
        end
    end

    end
    C2Q_mean=Q_mean;
    C2vehperhour=vehperhour;
    C2Avg_Speed_mean=Avg_Speed_mean;
    C2TmInNetTot_mean=TmInNetTot_mean;
    C2TSI_Number=TSI_Number;

    C2NFD_count_net=count_net;
    C2NFD_count_arr=count_arr;

    result=strcat('results\Nnw2\case_2\case_2','.mat');
    save(result,'NSIL','NSISC','C2Q_mean', 'C2vehperhour', '
        C2Avg_Speed_mean', 'C2TmInNetTot_mean','C2TSI_Number','
        C2y_25','C2x_25','C2Q_sum_25','C2NFD_count_net','
        C2NFD_count_arr');
```

```
%Max-weight algorithm with a noise level of 10% and 25%

noise=0.1; %change it to 0.25 for changing it to the case of
```

```
    25% noise level
Nnw2_case=310
%Nnw2_case=325
% Create/Verify folders
if exist('results/Nnw2/case_3/case_325','dir')==0
    mkdir('results/Nnw2/case_3/case_325');
end

if exist('results/Nnw2/case_3/case_310','dir')==0
    mkdir('results/Nnw2/case_3/case_310');
end

inflow_values=[0.5,1.0,1.5,2.0,2.5,3.0];
l=0;

Q_mean=size(1,size(inflow_values,2));
vehperhour=size(1,size(inflow_values,2));
TmInNetTot_mean=size(1,size(inflow_values,2));
Avg_Speed_mean=size(1,size(inflow_values,2));

TSI_vehdeparted=size(1,size(inflow_values,2));
TSI_Number=size(1,size(inflow_values,2));

count_net=cell(1,size(inflow_values,2));
count_arr=cell(1,size(inflow_values,2));

%Comment 10 percent noise or 25 percent parameters as per
    the case
C310Q_sum_25 =[0]; C310x_25 =[0]; C310y_25 =[0];
%C325Q_sum_25 =[0]; C325x_25 =[0]; C325y_25 =[0];

for inflow=inflow_values
l=l+1;
% SIMULATION DATA

Tsig=20;                              % Control Period Time
seconds=3600;                         % 3600 (simulation time in
    seconds)
sampling_time=5;

% INITIALIZATION

load('nine_network_sc_II.mat')      % Load SC Network
    Properties
load('nine_lanes_table_sc_II.mat') % Load SC Network
```

```
    Properties
45
46 vissim_com=actxserver('Vissim.Vissim.1100');
47 vissim_com.LoadNet([pwd '\Nnw2.inpx']);
48
49 sim=vissim_com.Simulation;
50 vnet=vissim_com.Net;
51 scs=vnet.SignalControllers;
52 links=vnet.Links;
53 demand_matrix1=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(1).Matrix;
54 demand_matrix2=vnet.DynamicAssignment.DynAssignDemands.
       ItemByKey(2).Matrix;
55
56 TSI_max_phases=max(NSISC.Phases);
57
58 ind=ones(1,size(NSISC,1));
59 prev_ind=ones(1,size(NSISC,1));
60
61 Volume=zeros(size(NSIL,1),seconds/sampling_time);
62 Speed=zeros(size(NSIL,1),seconds/sampling_time);
63 Avg_Speed=zeros(size(NSIL,1),seconds/sampling_time);
64 DelayTm=zeros(size(NSIL,1),seconds/sampling_time);
65 TmInNetTot=zeros(size(NSIL,1),seconds/sampling_time);
66 Q=zeros(size(NSIL,1),seconds/sampling_time);
67 QN=zeros(size(NSIL,1),seconds/sampling_time);
68 NL=zeros(size(NSIL,1),seconds/sampling_time);
69
70 Number=strings(size(NSIL,1),seconds/sampling_time);
71 N="";
72 DZone=cell(size(NSIL,1),seconds/sampling_time);
73 OZone=cell(size(NSIL,1),seconds/sampling_time);
74 ND=strings(size(NSIL,1),seconds/sampling_time);
75
76 vehdeparted=zeros(1,seconds);
77
78 LinkID_list=NSIL{:,1}';
79 LaneID_list=NSIL{:,2}';
80
81 conversion_list=zeros(size(NSIL,1),max(NSISC.Phases));
82 for i=1:size(NSIL,1)
83     conversion_list(LinkID_list(i),LaneID_list(i))=i;
84 end
85
86 b=NSISC{:,1};
```

```
87
88  shs=cell(size(NSISC,1),TSI_max_phases);
89  for i=1:size((NSISC{:,1}),1)
90      for j= 1:NSISC.Phases(i)
91          shs{i,j}=scs.ItemByKey(b(i)).SGs.GetAll{j};
92      end
93  end
94
95  % Record of State of Signal Heads
96  Signal_heads_states=ones(size(NSISC,1),TSI_max_phases,
        seconds);
97  for k=1:seconds
98   for j=1:size(NSISC,1)
99     if NSISC{j,2}~=TSI_max_phases
100      for i=NSISC{j,2}+1:TSI_max_phases
101          Signal_heads_states(j,i,k)=0;
102       end
103     end
104   end
105 end
106
107 Link_length=cell(1,size(NSIL,1));
108 Link_name=cell(1,size(NSIL,1));
109 for i=1:size(NSIL,1)
110     Link_length{i}=links.ItemByKey(LinkID_list(i)).AttValue(
          'Length2D');%Link length in m
111     Link_name{i}=links.ItemByKey(LinkID_list(i)).AttValue('
          Name');%Link length in m
112 end
113
114 for row=1:demand_matrix1.RowCount()
115     for col=1:demand_matrix1.ColCount()
116             matrix_cell=demand_matrix1.GetValue(row,col)*
                  inflow;
117             demand_matrix1.SetValue(row,col,matrix_cell)
118     end
119 end
120
121 for row=1:demand_matrix2.RowCount()
122     for col=1:demand_matrix2.ColCount()
123         matrix_cell=demand_matrix2.GetValue(row,col)*inflow;
124         demand_matrix2.SetValue(row,col,matrix_cell)
125     end
126 end
127
```

```
128 %% SIMULATION
129 sim.set('AttValue','SimRes',1);
130 eval=vissim_com.Evaluation;
131 set(vissim_com.Graphics,'AttValue','Quickmode',1);
132
133 for k=1:seconds
134     vissim_com.Simulation.RunSingleStep;                    % Run next Step (1 second)
135     vehdeparted(k)=vnet.Vehicles.GetDeparted.Count;
136     %-------------------------------------------------------------------------

137     % Initialize Signal Controllers per k==1s
138     if k==1
139         message=strcat('Progress: ',num2str(k/seconds*100,4),'% [',num2str(k),'s]');
140         disp(message)
141          for j=1:size(NSISC,1)
142              for i=1:NSISC{1,2}
143                  set(scs.ItemByKey(b(j)).SGs.ItemByKey(i),'AttValue','ContrByCOM',1);
144                  %The number 11 is the controller number. Would change with the selection of intersection.
145                  set(scs.ItemByKey(b(j)).SGs.ItemByKey(i),'AttValue','State',1);
146                  if i==1
147                      set(scs.ItemByKey(b(j)).SGs.ItemByKey(i),'AttValue','State',3);
148                  end
149              end
150          end
151     end
152
153     %-------------------------------------------------------------------------

154
155     if mod(k,sampling_time)==0
156
157         %use the code below to call out all the selected lanes from the table
158          for i=1:size(NSIL,1)
159              Vehicle_Attributes = cell2mat(links.ItemByKey(NSIL{i,1}).Lanes.ItemByKey(NSIL{i,2}).Vehs.
```

```
        GetAll);
160
161     if(size(Vehicle_Attributes,1)>0)
162         DZone{i,k/sampling_time}=size(1,size(
                Vehicle_Attributes,1));
163
164       for z=1:size(Vehicle_Attributes,1)
165         Volume(i,k/sampling_time)= Volume(i,k/
              sampling_time)+1;
166
167           dtm=Vehicle_Attributes(z).AttValue('
                DelayTm');
168           tin=Vehicle_Attributes(z).AttValue('
                TmInNetTot');
169           spd=Vehicle_Attributes(z).AttValue('
                Speed');
170           no=num2str(Vehicle_Attributes(z).
                AttValue('No'));
171
172         Number(i,k/sampling_time)=append(no,'-',
              Number(i,k/sampling_time));
173
174          DZone{i,k/sampling_time}(z)=str2double(
               Vehicle_Attributes(z).AttValue('
               DestZone'));
175          if(~isempty(DZone{i,k/sampling_time}(z)
               ))
176            ND(i,k/sampling_time)=append(no,'-',
                 num2str(DZone{i,k/sampling_time}(
                 z)),'/',ND(i,k/sampling_time));%
                 DZone{i,k/sampling_time}(z);%no;
177          end
178
179         Speed(i,k/sampling_time)=spd+Speed(i,k/
              sampling_time);
180         if spd<4
181           Q(i,k/sampling_time)=Q(i,k/
                sampling_time)+1;
182         end
183         DelayTm(i,k/sampling_time)=DelayTm(i,k/
              sampling_time)+dtm;
184         TmInNetTot(i,k/sampling_time)=TmInNetTot
              (i,k/sampling_time)+tin;
185       end
186     end
```

```
187                 if(Speed(i,k/sampling_time)~=0)
188                     Avg_Speed(i,k/sampling_time)=Speed(i,k/
                            sampling_time)/size(NSIL,1);
189                 end
190
191                 % Noise Level: avg length car = 5m
192                 % Number of maximum cars in a lane * % of error
193                 NL(i,k/sampling_time)=round(Link_length{i}/5*
                        noise);
194                 % Queue with noise
195                 QN(i,k/sampling_time)=max(Q(i,k/sampling_time)+
                        randi([-NL(i,k/sampling_time) NL(i,k/
                        sampling_time)],1,1),0);
196
197             end
198
199         end
200
201         %-------------------------------------------------------------------------
202         % BACK PRESSURE Algorithm
203
204         if mod(k,Tsig)==0 && k<seconds
205
206
207             for j=1:size(NSISC,1)
208
209                 % Pressure release calculation
210
211                 pa=zeros(1,NSISC{j,2});
212                 for phase_sc=1:NSISC{j,2}
213
214                     for option_sc=1:size(NSISC{j,((phase_sc-1)
                            *4+4)}{:},1)
215                         queue_cars_sc_in=0;
216                         queue_cars_sc_out=0;
217                         for option_sc_2=1:size(NSISC{j,((
                                phase_sc-1)*4+4)}{:},2)    % IN Links
                                and Lanes + queue
218
219                             link_sc_in = NSISC{j,((phase_sc-1)
                                    *4+4)}{:}(option_sc,option_sc_2);
220                             lane_sc_in = NSISC{j,((phase_sc-1)
                                    *4+5)}{:}(option_sc,option_sc_2);
```

```
221
222                     if link_sc_in ~= 0 || lane_sc_in ~=
                            0
223                         queue_cars_sc_in =
                                queue_cars_sc_in + QN(
                                conversion_list(link_sc_in,
                                lane_sc_in),k/sampling_time);
224                     end
225                 end
226
227                 for option_sc_2=1:size(NSISC{j,((
                        phase_sc-1)*4+6)}{:},2)    % OUT Links
                        and Lanes + queue
228
229                     link_sc_out = NSISC{j,((phase_sc-1)
                            *4+6)}{:}(option_sc,option_sc_2);
230                     lane_sc_out = NSISC{j,((phase_sc-1)
                            *4+7)}{:}(option_sc,option_sc_2);
231
232                     if link_sc_out ~= 0 || lane_sc_out
                            ~= 0
233                         queue_cars_sc_out =
                                queue_cars_sc_out + QN(
                                conversion_list(link_sc_out,
                                lane_sc_out),k/sampling_time)
                                ;
234                     end
235                 end
236
237                 pa(phase_sc) = pa(phase_sc) + max(
                        queue_cars_sc_in - queue_cars_sc_out
                        , 0 );  % PA calc
238
239             end
240         end
241
242         [maxx,ind(j)]=max(pa);
243         % use of ind(b(j)) and ind(j) are
                interchangeable. The value
244         % under parantheses depends on whether the whole
                network is
245         % controlled or just one particular intersection
246         s=find(maxx-pa==0);
247         if length(s)>1
248             ind(j)=s(randi([1 length(s)],1));%ind (b(j))
```

```
                    and ind(j)
249             end
250
251             % modify the signal heads
252             if ind(j)~=prev_ind(j)

                %ind (b(j)) and ind(j)
253              %fprintf('Algorithm used')
254              set(shs{j,ind(j)},'AttValue','State',3);
                                        % Green Light
                        %ind (b(j)) and ind(j)
255              set(shs{j,prev_ind(j)},'AttValue','State',1)
                    ;                   % Red Light
                   %ind (b(j)) and ind(j)
256              prev_ind(j)=ind(j);

                   %ind (b(j)) and ind(j)
257             end
258
259         end       %end of junctions
260
261
262     end
263
264     for j=1:size(ind,2)
265         Signal_heads_states(j,ind(j),k)=3;
266     end
267 end
268
269 %Result section for DZone and OZone, irrelevant in our case.
270 DZone;
271 Dzone_res=[];
272 for i=1:size(NSIL,1)
273     for j=1:seconds/sampling_time
274         if(~isempty(DZone{i,j}))
275             for z=1:size(DZone{i,j},2)
276                 tmp=DZone{i,j}(z);
277                 Dzone_res=[Dzone_res,tmp];
278             end
279         end
280     end
281 end
282 TSI_Dzone_edges=unique(Dzone_res);
283 TSI_Dzone_counts=(histc(Dzone_res(:),TSI_Dzone_edges))';
284
```

```
285 %The unique IDs of vehicles in the network is calculated
    below. Further
286 %stored in TSINumber.
287 Number;
288 Num_tmp="";
289 for i=1:size(NSIL,1)
290     for j=1:seconds/sampling_time
291         Num_tmp=append(Num_tmp,Number(i,j));
292     end
293 end
294 Num_tmp;
295 Num_tmp_2=split(Num_tmp,"-");
296 Num_tmp_3=str2double(unique(Num_tmp_2));
297 Num_tmp_3=Num_tmp_3(~isnan(Num_tmp_3));
298
299 Avg_Speed_mean(l)=mean(sum(Avg_Speed))/size(Avg_Speed,1);
300 TmInNetTot_mean(l)=mean(sum(TmInNetTot)/size(TmInNetTot,1));
301
302 Q_temp=zeros(size(Q,1),seconds/sampling_time);
303 Q_sum=zeros(1,seconds/sampling_time);
304 for i=1:seconds/sampling_time
305     for j=1:size(Q,1)
306         Q_temp(j,i)=Q(j,i);
307     end
308     Q_sum(i)=sum(Q_temp(:,i));
309
310 end
311
312 %We will get a plot of the sum of queues over a period of
    one hour, inflow
313 %set to the last inflow value.
314 plot(sampling_time:sampling_time:seconds,Q_sum);
315 Q_avg=sum(Q_temp)/size(Q,1);
316 Q_mean(l)=mean(Q_avg);
317 Q_avg_max=mean(Q_avg)+std(Q_avg)*3.5;
318 Q_avg_min=mean(Q_avg)-std(Q_avg)*3.5;
319
320 interval=0.1;
321 x=[Q_avg_min:interval:Q_avg_max];
322 y=normpdf(x,mean(Q_avg),std(Q_avg));
323
324 %Parameters measured over every inflow value as an array
    element.
325 vehperhour(l)=sum(vehdeparted);
326 %vehperhour is the count of departed vehicles from the
```

```
    previously marked
%parking space during each simulation.
TSI_Number(l)=size(Num_tmp_3,1);
%TSI_Number is the sum of unique vehicles present during
    each inflow rate.

%
    ------------------------------------------------------------------------


%Aggregated sum of queues and their mean's normal
    distribution-inflow 2,5
if(inflow==2.5)
      C310Q_sum_25=Q_sum;
      C310x_25=x;
      C310y_25=y;
%       C325Q_sum_25=Q_sum;
%       C325x_25=x;
%       C325y_25=y;
end

%
    ------------------------------------------------------------------------


%NFD parameters for all the inflow values are grouped
    together here and
%stored as array values. Hence, for our calculation we
    extract all inflow
%values based fundamental diagram.

%Number;
temp="";
List=strings(1,seconds/sampling_time);
for i=1:seconds/sampling_time
     for j=1:size(NSIL,1)
          temp=append(temp,Number(j,i));
     end
     List(1,i)=temp;
     temp="";
end
List2=cell(1,seconds/sampling_time);
sz=zeros(1,seconds/sampling_time);
for i=1:seconds/sampling_time
    %With the help of this expression, we get the ID of
        vehicles involved
    %every 5 mins.
```

```
362         List2{i}=sort(str2double(split(List{i},"-")));
363         sz(i)=size(List2{i},1);
364     end
365
366     %count tells us the number of vehicles that have left from
        the network or
367     %in other words have reached their destination.
368     k=[];
369     ID={};
370     for i=1:seconds/sampling_time
371
372         for m=1:i
373           k=[k,List2{m}'];
374         end
375         n=rmmissing(sort(k(:)'));
376         g=rmmissing(unique(k));
377         t=histc(n,g);
378
379         for o=1:size(t,2)
380             ID{i,o}=[g(o),t(o)];
381         end
382
383         k=[];
384     end
385     %ID;
386
387     idx=[];
388     Last_ID={};
389     for i=1:size(ID,2)
390         Last_ID{i}=ID{size(ID,1),i};
391     end
392
393     count=1;
394     moments=zeros(1,size(ID,2));
395     for i=1:size(ID,2)
396         for j=1:size(ID,1)
397             if(~isempty(ID{j,i}))
398                 if(ID{j,i}(2)==Last_ID{1,i}(2))
399                     moments(count)=j+1;
400                     count=count+1;
401                     break;
402                 end
403             end
404         end
405     end
```

```
406
407 count_arr{l}=zeros(1,60);
408 time_stamp=[1:12:size(ID,1)];
409 for i=1:size(moments,2)
410     for j=1:size(time_stamp,2)
411         if(moments(1,i)<time_stamp(1,j))
412             count_arr{l}(1,j)=count_arr{l}(1,j)+1;
413             break;
414         end
415     end
416 end
417
418 count_net{l}=zeros(1,60);
419 for i=1:size(sz,2)
420     if(mod(i,12)==0)
421         i;
422         count_net{l}(i/12)=sz(i);
423     end
424 end
425 end
426
427 C310Q_mean=Q_mean;
428 C310vehperhour=vehperhour;
429 C310Avg_Speed_mean=Avg_Speed_mean;
430 C310TmInNetTot_mean=TmInNetTot_mean;
431 C310TSI_Number=TSI_Number;
432
433 % C325Q_mean=Q_mean;
434 % C325vehperhour=vehperhour;
435 % C325Avg_Speed_mean=Avg_Speed_mean;
436 % C325TmInNetTot_mean=TmInNetTot_mean;
437 % C325TSI_Number=TSI_Number;
438
439
440 C310NFD_count_net=count_net;
441 C310NFD_count_arr=count_arr;
442 % C325NFD_count_net=count_net;
443 % C325NFD_count_arr=count_arr;
444
445 result=strcat('results\Nnw2\case_3\case_3', num2str(noise
    *100),'\case_3', num2str(noise*100),'.mat');
446 save(result,'NSIL','NSISC','C310Q_mean', 'C310vehperhour', '
    C310Avg_Speed_mean', 'C310TmInNetTot_mean','
    C310TSI_Number','C310y_25','C310x_25','C310Q_sum_25','
    C310NFD_count_net','C310NFD_count_arr');
```

```
447 %save(result,'NSIL','NSISC','C325Q_mean', 'C325vehperhour',
    'C325Avg_Speed_mean', 'C325TmInNetTot_mean','
    C325TSI_Number','C325y_25','C325x_25','C325Q_sum_25','
    C325NFD_count_net','C325NFD_count_arr');
```

C.3 Plotting

All the parameters mentioned under the '.mat' file are then plotted. Individual as well as collective plotting of data points is done via this script.

```
1  clear title;
2  close all;
3
4  %Let's compare how an unsaturated network would behave when
       the inflow rate
5  %is set to 2,5.
6  %Import Q_sum when inflow rate is 2,5 in each case along
       with the normal
7  %distribution of the Q_mean for that particular network
8  sampling_time=5;
9  seconds=3600;
10 cases_values=[1,2,310,325];
11
12 data_directory='results\Nnw2';
13 output_directory='results\Nnw2\figures';
14 if exist(output_directory,'dir')==0
15     mkdir(output_directory);
16 end
17 if exist(strcat(output_directory,'\','normal_dist\'),'dir')
       ==0
18     mkdir(strcat(output_directory,'\','normal_dist\'));
19 end
20 if exist(strcat(output_directory,'\','Q_sum_25\'),'dir')==0
21     mkdir(strcat(output_directory,'\','Q_sum_25\'));
22 end
23 if exist(strcat(output_directory,'\','NFD\'),'dir')==0
24     mkdir(strcat(output_directory,'\','NFD\'));
25 end
26 if exist(strcat(output_directory,'\','General\'),'dir')==0
27     mkdir(strcat(output_directory,'\','General\'));
28 end
29 %%
30 for num_case=cases_values
31     if or(num_case==310,num_case==325)
```

```
32          input_file_name=strcat(data_directory,'\case_3','\',
                'case_',num2str(num_case),'\','case_',num2str(
                num_case),'.mat');
33
34          %Inflow 2,5 | Aggregated sum of queues | Normal
                distribution
35          load(input_file_name,strcat('C',num2str(num_case),'
                Q_sum_25'))
36          load(input_file_name,strcat('C',num2str(num_case),'
                x_25'))
37          load(input_file_name,strcat('C',num2str(num_case),'
                y_25'))
38
39          %NFD information contained in cell format that has
                vehicle arriving
40          %and vehicle density every minute. It contains data
                for all inflow
41          %multipliers.
42          load(input_file_name,strcat('C',num2str(num_case),'
                NFD_count_net'))
43          load(input_file_name,strcat('C',num2str(num_case),'
                NFD_count_arr'))
44
45          %average speed mean, queue mean, time spent in the
                network,
46          %unique vehicles in the network.
47          load(input_file_name,strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'Q_mean'),
                strcat('C',num2str(num_case),'TmInNetTot_mean'),
                strcat('C',num2str(num_case),'TSI_Number'))
48          load(input_file_name,strcat('C',num2str(num_case),'
                Avg_Speed_mean'),strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                TSI_Number'))
49          load(input_file_name,strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                TSI_Number'))
50          load(input_file_name,strcat('C',num2str(num_case),'
                TSI_Number'),strcat('C',num2str(num_case),'Q_mean
                '),strcat('C',num2str(num_case),'TmInNetTot_mean'
                ),strcat('C',num2str(num_case),'TSI_Number'))
51      else
```

```
52          input_file_name=strcat(data_directory,'\','case_',
                num2str(num_case),'\','case_',num2str(num_case),'
                .mat');
53
54          %Inflow 2,5 | Aggregated sum of queues | Normal
                distribution
55          load(input_file_name,strcat('C',num2str(num_case),'
                Q_sum_25'))
56          load(input_file_name,strcat('C',num2str(num_case),'
                x_25'))
57          load(input_file_name,strcat('C',num2str(num_case),'
                y_25'))
58
59          %NFD information contained in cell format that has
                vehicle arriving
60          %and vehicle density every minute. It contains data
                for all inflow
61          %multipliers.
62          load(input_file_name,strcat('C',num2str(num_case),'
                NFD_count_net'))
63          load(input_file_name,strcat('C',num2str(num_case),'
                NFD_count_arr'))
64
65          %average speed mean, queue mean, time spent in the
                network,
66          %unique vehicles in the network.
67          load(input_file_name,strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'Q_mean'),
                strcat('C',num2str(num_case),'TmInNetTot_mean'),
                strcat('C',num2str(num_case),'TSI_Number'))
68          load(input_file_name,strcat('C',num2str(num_case),'
                Avg_Speed_mean'),strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                TSI_Number'))
69          load(input_file_name,strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                Q_mean'),strcat('C',num2str(num_case),'
                TmInNetTot_mean'),strcat('C',num2str(num_case),'
                TSI_Number'))
70          load(input_file_name,strcat('C',num2str(num_case),'
                TSI_Number'),strcat('C',num2str(num_case),'Q_mean
                '),strcat('C',num2str(num_case),'TmInNetTot_mean'
                ),strcat('C',num2str(num_case),'TSI_Number'))
71      end
```

```
72 end
73 %%
74 %
75
76 %Plots to compare sum of queues
77 hold on
78 plot(sampling_time:sampling_time:seconds,C1Q_sum_25);
79 lgd=legend('case 1: Fixed and cyclic Signal Heads');
80 lgd.Location='northwest';
81 plot(sampling_time:sampling_time:seconds,C2Q_sum_25,'
      DisplayName','case 2: BP Control without noise');
82 plot(sampling_time:sampling_time:seconds,C310Q_sum_25,'
      DisplayName','case 310: BP Control, 10% noise')
83 %plot(sampling_time:sampling_time:seconds,C410Q_sum_25,'
      DisplayName','case 410: BP Control, 10% noise, filtered')
      ;
84 hold off
85 xlabel('Time [s]');
86 ylabel('Sum of Queues [Vehicles]');
87 title('Aggregated queue length  for our network');
88
89 saveas(gcf,strcat(output_directory,'\Q_sum_25\','
      Q_sum_25_noise10'),'fig');
90 saveas(gcf,strcat(output_directory,'\Q_sum_25\','
      Q_sum_25_noise10'),'jpg');
91 close all;
92 %%
93
94 hold on
95 plot(sampling_time:sampling_time:seconds,C1Q_sum_25);
96 lgd=legend('case 1: Fixed and cyclic Signal Heads');
97 lgd.Location='northwest';
98 plot(sampling_time:sampling_time:seconds,C2Q_sum_25,'
      DisplayName','case 2: BP Control without noise');
99 plot(sampling_time:sampling_time:seconds,C325Q_sum_25,'
      DisplayName','case 325: BP Control, 25% noise');
100 %plot(sampling_time:sampling_time:seconds,C425Q_sum_25,'
      DisplayName','case 425: BP Control, 25% noise, filtered')
      ;
101 hold off
102 xlabel('Time [s]');
103 ylabel('Sum of Queues [Vehicles]');
104 title('Aggregated queue length  for our network');
105
```

```
106 saveas(gcf,strcat(output_directory,'\Q_sum_25\','
    Q_sum_25_noise25'),'fig');
107 saveas(gcf,strcat(output_directory,'\Q_sum_25\','
    Q_sum_25_noise25'),'jpg');
108 close all;
109 %%
110 %Plots to compare normal distribution of queue lengths
111 hold on
112 plot(C1x_25,C1y_25);
113 lgd=legend('case 1: Fixed and cyclic Signal Heads');
114 lgd.Location='southeast';
115 plot(C2x_25,C2y_25,'DisplayName','case 2: BP Control without
     noise');
116 plot(C310x_25,C310y_25,'DisplayName','case 310: BP Control,
    10% noise');
117 %plot(C410x_25,C410y_25,'DisplayName','case 410: BP Control,
     10% noise, filtered');
118 hold off
119 xlabel('Queue mean [vehicles]')
120 title('Normal dist of queue length for our network');
121
122 saveas(gcf,strcat(output_directory,'\normal_dist\','
    NPDF25_noise10'),'fig');
123 saveas(gcf,strcat(output_directory,'\normal_dist\','
    NPDF25_noise10'),'jpg');
124 close all;
125 %%
126 hold on
127 plot(C1x_25,C1y_25);
128 lgd=legend('case 1: Fixed and cyclic Signal Heads');
129 lgd.Location='southeast';
130 plot(C2x_25,C2y_25,'DisplayName','case 2: BP Control without
     noise');
131 plot(C325x_25,C325y_25,'DisplayName','case 325: BP Control,
    25% noise');
132 %plot(C425x_25,C425y_25,'DisplayName','case 425: BP Control,
     25% noise, filtered');
133 hold off
134 xlabel('Queue mean [vehicles]')
135 title('Normal dist of queue length for our network');
136
137 saveas(gcf,strcat(output_directory,'\normal_dist\','
    NPDF25_noise25'),'fig');
138 saveas(gcf,strcat(output_directory,'\normal_dist\','
    NPDF25_noise25'),'jpg');
```

```
close all;
%%
%Plot to compare NFD
% hold on
plot([C1NFD_count_net{:}],[C1NFD_count_arr{:}],'LineStyle',"
    none","Marker",".")
lgd=legend('case 1: Fixed and cyclic Signal Heads');
lgd.Location='northwest';
% plot([C2NFD_count_net{:}],[C2NFD_count_arr{:}],'LineStyle
    ',"none","Marker",".",'DisplayName','case 2: BP Control
    without noise')
% plot([C310NFD_count_net{:}],[C310NFD_count_arr{:}],'
    LineStyle',"none","Marker",".",'DisplayName','case 310:
    BP Control, 25% noise')
% %plot([C410NFD_count_net{:}],[C410NFD_count_arr{:}],'
    LineStyle',"none","Marker",".",'DisplayName','case 425:
    BP Control, 25% noise, filtered');
% hold off
ylim([0,100])
xlabel('Vehicles in the network (density)')
ylabel('Flow (vehicles/min)')
title('Network flow diagram');
%
% saveas(gcf,strcat(output_directory,'\NFD\','NFD_noise10')
    ,'fig');
% saveas(gcf,strcat(output_directory,'\NFD\','NFD_noise10')
    ,'jpg');
% close all;
%%

hold on
plot([C1NFD_count_net{:}],[C1NFD_count_arr{:}],'LineStyle',"
    none","Marker",".")
lgd=legend('case 1: Fixed and cyclic Signal Heads');
lgd.Location='northwest';
plot([C2NFD_count_net{:}],[C2NFD_count_arr{:}],'LineStyle',"
    none","Marker",".",'DisplayName','case 2: BP Control
    without noise')
plot([C325NFD_count_net{:}],[C325NFD_count_arr{:}],'
    LineStyle',"none","Marker",".",'DisplayName','case 325:
    BP Control, 25% noise')
%plot([C425NFD_count_net{:}],[C425NFD_count_arr{:}],'
    LineStyle',"none","Marker",".",'DisplayName','case 425:
    BP Control, 25% noise, filtered');
hold off
```

```
168 ylim([0,100])
169 xlabel('Vehicles in the network (density)')
170 ylabel('Flow (vehicles/min)')
171 title('Network flow diagram');
172
173 saveas(gcf,strcat(output_directory,'\NFD\','NFD_noise25'),'
    fig');
174 saveas(gcf,strcat(output_directory,'\NFD\','NFD_noise25'),'
    jpg');
175 close all;
176 %%
177 % General parameters.
178
179 %Plot to compare mean of queues
180 hold on
181 plot(C1TSI_Number,C1Q_mean);
182 lgd=legend('case 1: Fixed and cyclic Signal Heads');
183 lgd.Location='northwest';
184 plot(C2TSI_Number,C2Q_mean,'DisplayName','case 2: BP Control
     without noise');
185 plot(C310TSI_Number,C310Q_mean,'DisplayName','case 310: BP
    Control, 10% noise');
186 %plot(C410TSI_Number,C410Q_mean,'DisplayName','case 410: BP
    Control, 10% noise, filtered');
187 hold off
188 xlabel('Inflow [vehicles/hour]');
189 ylabel('Average queue length [Vehicles]');
190 title('Aggregated queue length  for our network');
191
192 saveas(gcf,strcat(output_directory,'\General\','
    Q_avg_noise10'),'fig');
193 saveas(gcf,strcat(output_directory,'\General\','
    Q_avg_noise10'),'jpg');
194 close all;
195 %%
196 hold on
197 plot(C1TSI_Number,C1Q_mean);
198 lgd=legend('case 1: Fixed and cyclic Signal Heads');
199 lgd.Location='northwest';
200 plot(C2TSI_Number,C2Q_mean,'DisplayName','case 2: BP Control
     without noise');
201 plot(C325TSI_Number,C325Q_mean,'DisplayName','case 325: BP
    Control, 25% noise')
202 %plot(C425TSI_Number,C425Q_mean,'DisplayName','case 425: BP
    Control, 10% noise, filtered');
```

```
203 hold off
204 xlabel('Inflow [vehicles/hour]');
205 ylabel('Average queue length [Vehicles]');
206 title('Aggregated queue length  for our network');
207
208 saveas(gcf,strcat(output_directory,'\General\','
    Q_avg_noise25'),'fig');
209 saveas(gcf,strcat(output_directory,'\General\','
    Q_avg_noise25'),'jpg');
210 close all;
211 %%
212 %Plot to compare average speed
213 hold on
214 plot(C1TSI_Number,C1Avg_Speed_mean);
215 lgd=legend('case 1: Fixed and cyclic Signal Heads');
216 lgd.Location='northwest';
217 plot(C2TSI_Number,C2Avg_Speed_mean,'DisplayName','case 2: BP
     Control without noise');
218 plot(C310TSI_Number,C310Avg_Speed_mean,'DisplayName','case
    310: BP Control, 10% noise');
219 %plot(C410TSI_Number,C410Avg_Speed_mean,'DisplayName','case
    410: BP Control, 10% noise, filtered');
220 hold off
221 xlabel('Inflow [vehicles/hour]');
222 ylabel('Average speed [km/hour]');
223 title('Average speed');
224
225 saveas(gcf,strcat(output_directory,'\General\','
    Avg_speed_noise10'),'fig');
226 saveas(gcf,strcat(output_directory,'\General\','
    Avg_speed_noise10'),'jpg');
227 close all;
228 %%
229 hold on
230 plot(C1TSI_Number,C1Avg_Speed_mean);
231 lgd=legend('case 1: Fixed and cyclic Signal Heads');
232 lgd.Location='northwest';
233 plot(C2TSI_Number,C2Avg_Speed_mean,'DisplayName','case 2: BP
     Control without noise');
234 plot(C325TSI_Number,C325Avg_Speed_mean,'DisplayName','case
    325: BP Control, 25% noise');
235 %plot(C425TSI_Number,C425Avg_Speed_mean,'DisplayName','case
    425: BP Control, 25% noise, filtered');
236 hold off
237 xlabel('Inflow [vehicles/hour]');
```

```
238 ylabel('Average speed [km/hour]');
239 title('Average speed');
240
241 saveas(gcf,strcat(output_directory,'\General\','
    Avg_speed_noise25'),'fig');
242 saveas(gcf,strcat(output_directory,'\General\','
    Avg_speed_noise25'),'jpg');
243 close all;
244 %%
245 %Plot to compare time spent in the network
246 hold on
247 plot(C1TSI_Number,C1TmInNetTot_mean);
248 lgd=legend('case 1: Fixed and cyclic Signal Heads');
249 lgd.Location='northwest';
250 plot(C2TSI_Number,C2TmInNetTot_mean,'DisplayName','case 2:
    BP Control without noise');
251 plot(C310TSI_Number,C310TmInNetTot_mean,'DisplayName','case
    310: BP Control, 10% noise');
252 %plot(C410TSI_Number,C410TmInNetTot_mean,'DisplayName','case
     410: BP Control, 10% noise, filtered');
253 hold off
254 xlabel('Inflow [vehicles/hour]');
255 ylabel('Average travel time [seconds]');
256 title('Time spent in the network');
257
258 saveas(gcf,strcat(output_directory,'\General\','
    Avg_travel_time_noise10'),'fig');
259 saveas(gcf,strcat(output_directory,'\General\','
    Avg_travel_time_noise10'),'jpg');
260 close all;
261 %%
262 hold on
263 plot(C1TSI_Number,C1TmInNetTot_mean);
264 lgd=legend('case 1: Fixed and cyclic Signal Heads');
265 lgd.Location='northwest';
266 plot(C2TSI_Number,C2TmInNetTot_mean,'DisplayName','case 2:
    BP Control without noise');
267 plot(C325TSI_Number,C325TmInNetTot_mean,'DisplayName','case
    325: BP Control, 25% noise');
268 %plot(C425TSI_Number,C425TmInNetTot_mean,'DisplayName','case
     425: BP Control, 25% noise, filtered');
269 hold off
270 xlabel('Inflow [vehicles/hour]');
271 ylabel('Average travel time [seconds]');
272 title('Time spent in the network');
```

```
273
274 saveas(gcf,strcat(output_directory,'\General\','
    Avg__travel_time_noise25'),'fig');
275 saveas(gcf,strcat(output_directory,'\General\','
    Avg__travel_time_noise25'),'jpg');
276 close all;
```

D Implementation Guide

1. Primary step

1a. Mandatory files

- Nnw_case_#.m (example: Nnw_case_3.m)
- nine_network_sc_II.mat
- nine_lanes_table_sc_II.mat
- PTV Vissim files (.inpx, etc)

Important note: *Nnw_case*3.*m* corresponds to cases 310 and 325. The other main simulation files, *Nnw_case*1.*m* and *Nnw_case*2.*m* are based on *Nnw_case*3.*m*. By commenting extensively on the *Nnw_case*1.*m* main script, we can omit repeating ourselves in the other two set of cases.

List of main simulation files:

- Nnw_case1.m : case 100
- Nnw_case2.m : case 200
- Nnw_case3.m : cases 310 and 325

1b. Network data

If new *nine_network_sc_II.mat* and *nine_lanes_table_sc_II.mat* need to be generated because the network has changed (if even as small as just one junction has to be used), the *conversion.m* (see appendix **??**) function has to be executed. The function converts it in *.csv* format table, as seen in appendix **??**, into these two files.

2. Launch the simulation

2a. Adjust parameters

1. Open the *Nnw_case#.m* file.
2. Make sure that *num_case* variable corresponds to the selected case #.

3. $inflow\_values$: Adjust the inflow values multiplier if you want individual case returns or if you want to increase or decrease the flow further.
4. $noise\_values$: Adjust the noise level values that will be simulated in case 3.
5. Along with it, check for 310 or 325 in the script. Whenever there is a noise level adjustment, peruse through the script and comment the case that is not being considered.

Optional:

- $Tsig$: control period time in seconds (phase duration).
- $seconds$: simulation time in seconds.
- $sampling\_time$: sampling period of the data, in seconds.

2b. Start the simulation

Once the main file is ready, the code can be executed (MATLAB RUN BUTTON). All the combinations of the $inflow\_values$ and $noise\_values$ will be simulated, one after another.

2c. Saved data

After each simulation, data is saved automatically in the $results\_Nnw2$ folder. The folder is further subdivided case-wise and plot-wise. Thus, after a run the values updated and plotted would overwrite the right set of information. The values used while doing this particular submission is stored in the subfolder final.

3. Plot results

3a. Required files

The script *Plotting.m* is an all encompassing file that can be used to plot:

- Sum of queues and do their simultaneous comparison.
- Normal distribution of queue lengths and to compare them.
- NFDs, both individually and cumulatively.
- General parameters: comparison of mean of queues, average speed and time spent in the network.

3b. Adjust parameters

List of parameters to be adjusted (it depends of the acquired data):

- *inflow_values*: simulated inflow values.
- *noise_values*: simulated noise values.
- *seconds*: simulated time in seconds.
- *sampling_time*: sampling period used during the simulation, in seconds.

There's also provision to change resolution time, but for a small network it can be overridden.

3c. Saved figures

Launch the desired scripts. All the generated figures are saved and sorted as per their nomenclature in the *figures* folder.